COLINGO C-10 USERS' MANUAL

VOLUME II

MAY 1968

COLINGO Project

Prepared for

AIR FORCE COMMAND AND MANAGEMENT SYSTEMS DIVISION
DEPUTY FOR COMMAND SYSTEMS
ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
L. G. Hanscom Field, Bedford, Massachusetts

Project 512V
Prepared by

THE MITRE CORPORATION
Bedford, Massachusetts

Contract AF19(628)-5165

COLINGO C-10 USERS' MANUAL

VOLUME II

MAY 1968

COLINGO Project

Prepared for

AIR FORCE COMMAND AND MANAGEMENT SYSTEMS DIVISION
DEPUTY FOR COMMAND SYSTEMS
ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
L. G. Hanscom Field, Bedford, Massachusetts

Project 512V
Prepared by

THE MITRE CORPORATION
Bedford, Massachusetts

Contract AF19(628)-5165

FOREWORD

This report was prepared by The MITRE Corporation, Bedford, Massachusetts, under Contract AF 19(628)-5165, Projects 504F, 512B, and 512V. Portions were written over the period 15 December 1965 to 23 February 1968 and provide a complete Users' Manual.

ESD Project Officer: Lt Col James L. Blilie, ESLFE.

REVIEW AND APPROVAL

This technical report has been reviewed and is approved.

JAMES L. BLILIE, Lt Col, USAF
Chief, Engineering Support Branch

ABSTRACT

The COLINGO C-10 Users' Manual, a combination of tutorial and reference material, is presented in two volumes. This volume contains information on machine configurations, procedures for operating and loading the system, a description of the C-10 general purpose macro facility (terses and actors), a guide to the STEP language, a set of instructions for preparing machine language procedures, and a list of system error messages.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Concluded)

LIST OF ILLUSTRATIONS

SECTION I

MACHINE CONFIGURATION

INTRODUCTION

"Machine Configuration" refers to (a) the type of equipment
(e.g., processors, memories, disks), (b) the number of each type,
and (c) the channel by which each is attached to the system.

The C-10 system has been designed to operate on a "minimum
configuration" of a 40K core, disk unit, typewriter, card-reader
and printer. Additional optional equipment may also be added.
Additional core and disk units increase the efficiency of the sys-
tem, but not its inherent capabilities. Other optional devices
which may be added to the system include tapes, displays, a remote
inquiry unit and a clock. The exact configuration is stated at
loading time.

MINIMUM CONFIGURATION

A minimum practical configuration for C-10 consists of an IBM 1411 central processor with the smallest available memory and associated console typewriter; also a card-reader punch, a printer and a disk storage unit. The following table describes these:

Table I

A C-10 Practical Configuration

ITEM	MODEL	CHARACTERISTICS
Central Processor	1411-3	40K characters, 4.5 us
Console Typewriter	1415	15cps, 80 characters/line
Disk	1301 or 1302 or 1311*	More storage, faster access than 1311
Card-reader punch	1402	
Printer	1403-2 or 1402-3	
Priority Alert**		

* If a 1311 disk is not used, some sort of tape must be used to load the system onto the disk being used.

** An interrupt system that provides an automatic branch to a fixed storage location when certain conditions of the I/O channels or devices occur.

OPTIONAL DEVICES

In addition to the minimal equipment, C-10 will accept many standard IBM devices such as tapes, a clock, etc. Facilities for a CCC display unit are also built into the system. The table below lists such optional items.

Table II

Optional Items

ITEM	MODEL
Magnetic Tape Units*	729-2,4,5 or 6 or 7330
Remote Inquiry Unit	
Clock	
CCC Display Unit	TRW

* Tapes are necessary in any routine that sorts files; otherwise they are optional.

SYSTEM ADAPTATION

The C-10 system is capable of restructuring itself to adapt to various configurations. For instance, if a central memory of 60K is being used, the C-10 system automatically assigns central memory space to some procedures that would be kept in disk storage with a 40K central memory. As the system operates, this space-sharing technique continually shuffles data and procedures from core to disk in an efficient way.

To be able to adapt to a particular configuration, the system must, at loading time, be told some information concerning the configuration. Of course, the system assumes the minimum apparatus mentioned previously, but it does not necessarily know where these are located, i.e., on what channel. The only initial assumption is that there is a typewriter on channel 1. Details pertaining to the declaration of machine configuration may be found in Section II.

DISK USAGE

C-10 is a disk-oriented system. It occupies only a portion of
the disk, the portion to be occupied being specified at loading time.
More than one disk may be used, with each different disk type attached
to a different channel.

Since C-10 does not occupy all available disk space, it has built-
in checks to assure that no "non-C-10" section of the disk is referenced.

C-10 is designed to allow use of the IBM 1301, 1302 and 1311 disks.
Although the 1301 and 1302 have more storage and faster access than a
1311, they require that one of the tapes listed in the optional config-
uration be used. With a 1311 this is not necessary, because a 1311
disk is detachable, and C-10 can be loaded at one installation and re-
installed at another. A 1301 or a 1302 disk, however, is not detachable;
hence, the system must be loaded onto them from tape.

VARIATIONS IN CONFIGURATION

Assuming a certain minimum configuration, it is not possible to increase the capabilities of the C-10 system. The efficiency and performance may be changed, however, by changing the configuration. The following are some variations in configuration and their effects.

(a) Addition of Optional Devices

These are the items listed in Table II, Optional Items. They are more likely to increase the performance, by providing different methods of operation, than to increase the speed of the system.

(b) Addition of More than One I/O Device

This allows for faster and more efficient input-output operations. Up to 28 of these peripheral devices may be attached on up to 2 channels.

(c) Larger Central Memory Size

The size of central memory has a large effect on the performance of the C-10 system. As many C-10 procedures as possible reside in central memory; the rest are allocated to bulk memory (the disk).

A space-sharing technique decides in an efficient way what procedures are to occupy central memory at any moment. Since any procedure that is invoked that is not in central memory must be fetched from the disk, the greater the number of procedures that can reside in central memory, the greater the efficiency.

(d) Changes in Disk Drive

This also has a large effect on the performance of
the system as a whole. As mentioned previously,
the 1301 and 1302 have larger storage capacity and
faster access, while the 1311 obviates the need for
tape devices.

LOADING AND OPERATING PROCEDURES

INTRODUCTION

The C-10 System may be stored on cards, tape, or disk. If it is stored on cards or tape, it must be loaded onto disk before it can be operated. This loading process is described in the loading instructions that follow.

All routines in C-10 are stored permanently on disk and may be brought into memory from disk by the relocatable subroutine controller. Loading programs directly into core memory from cards or tape is generally not meaningful. The allocation of the appropriate routines to core memory is performed automatically and does not concern the C-10 user.

After the subroutines have been allocated to disk, and initialization performed, the system is ready for operation as described in the operating procedures.*

The block diagram (Figure 1) shows the steps in loading from tape, card, or disk. Under the heading Disk are the steps which take place under normal use of the system. Thus the four shaded boxes are actually part of the operating procedures and are explained under operating procedures.

---

* If the reader is concerned only with operating the system (C-10 is stored on disk), then he should turn directly to OPERATING PROCEDURES (page 37).

TAPE

CARD

DISK

Operator loads system from TAPE

Operator loads system from CARDS.

Operator loads boot-strap card.

Operator makes changes in machine con-figuration if necessary.

System ready
System types
ENTER STARTING SUBROUTINE

System punches bootstrap card.

System types:
ENTER STARTING SUBROUTINE

System ready.
System types:
ENTER STARTING SUBROUTINE

To generate a new tape type 295

To generate a new tape, type 295

Load programs Type 237

SYSTEM LOADING AND MODIFYING FROM
TAPE
CARDS
DISK

Initialize Type 154

To generate a new tape, type STEP (SYSTAPE) △

Figure 1. Loading Steps

6

LOADING INSTRUCTIONS

Prior to loading disk from cards or tape, <u>core must be cleared</u>. After clearing core, the operator loads the cards into the card reader or the tape onto the tape drive. He then types an initial 11-character bootstrap instruction.

The C-10 loader receives control and, by interacting with the operator at the console, determines the hardware configuration of the computer onto which it has been loaded (the number of channels, type of disk, etc.).

Loading From Cards Onto Disk

Deck Setup for Loading from Cards
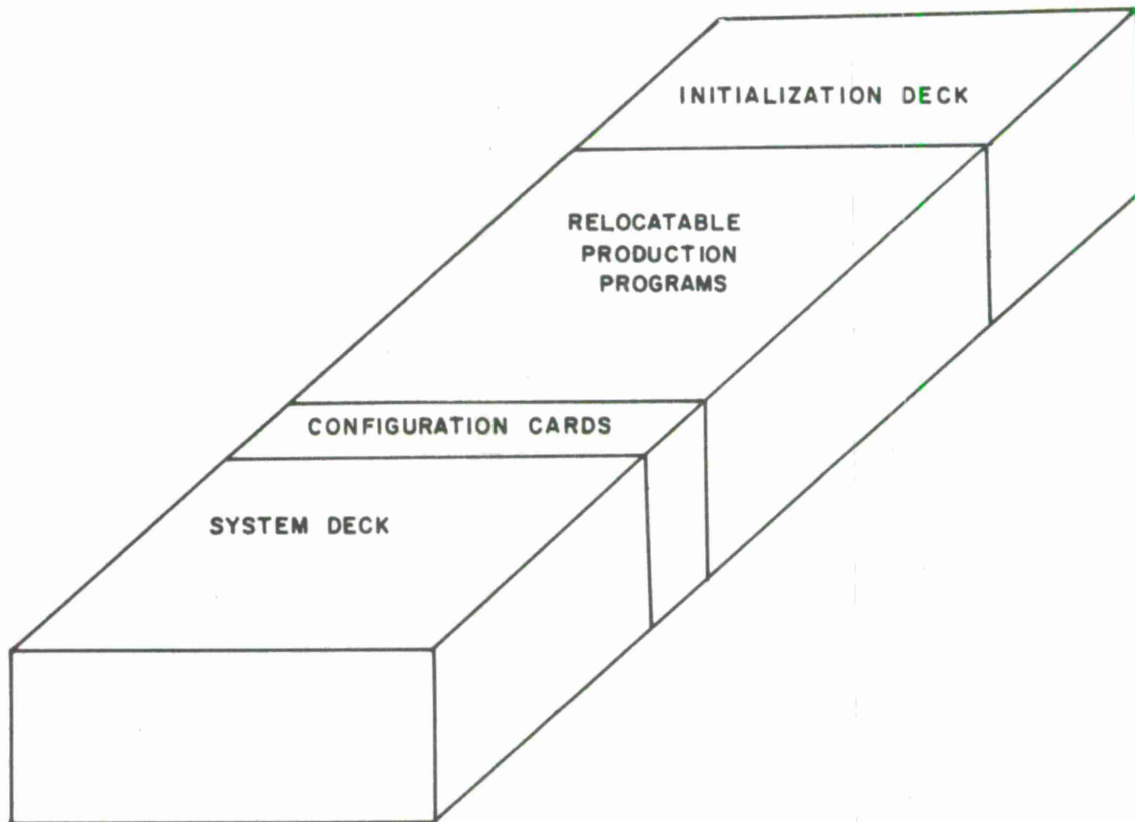
Figure 2. Loading from Cards

11

The initialization deck must be preceded by a card containing "DATA CARD." The D must start in Column 1.
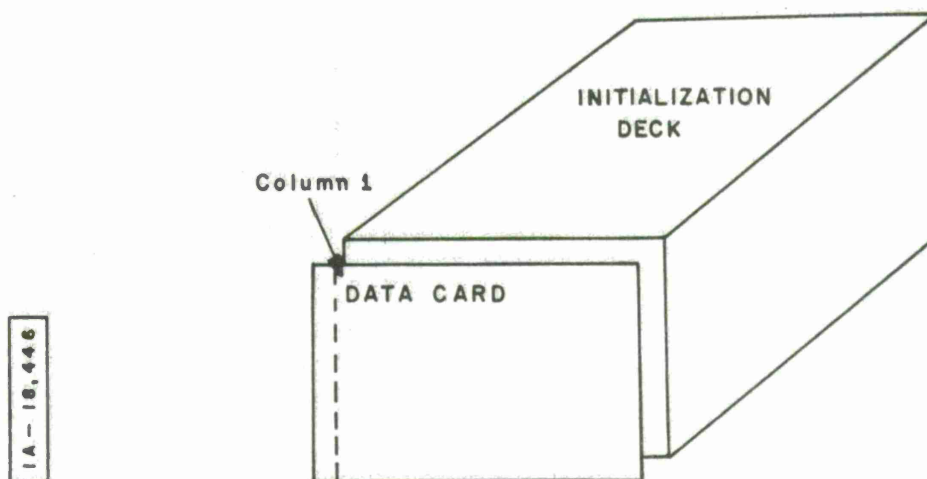
Figure 3. Data Card and Initialization Deck

The system deck consists of seven sections followed by
an END CARD. The system deck is supplied as a single unit; it
is not necessary for the deck to be constructed out of its
constituent pieces.
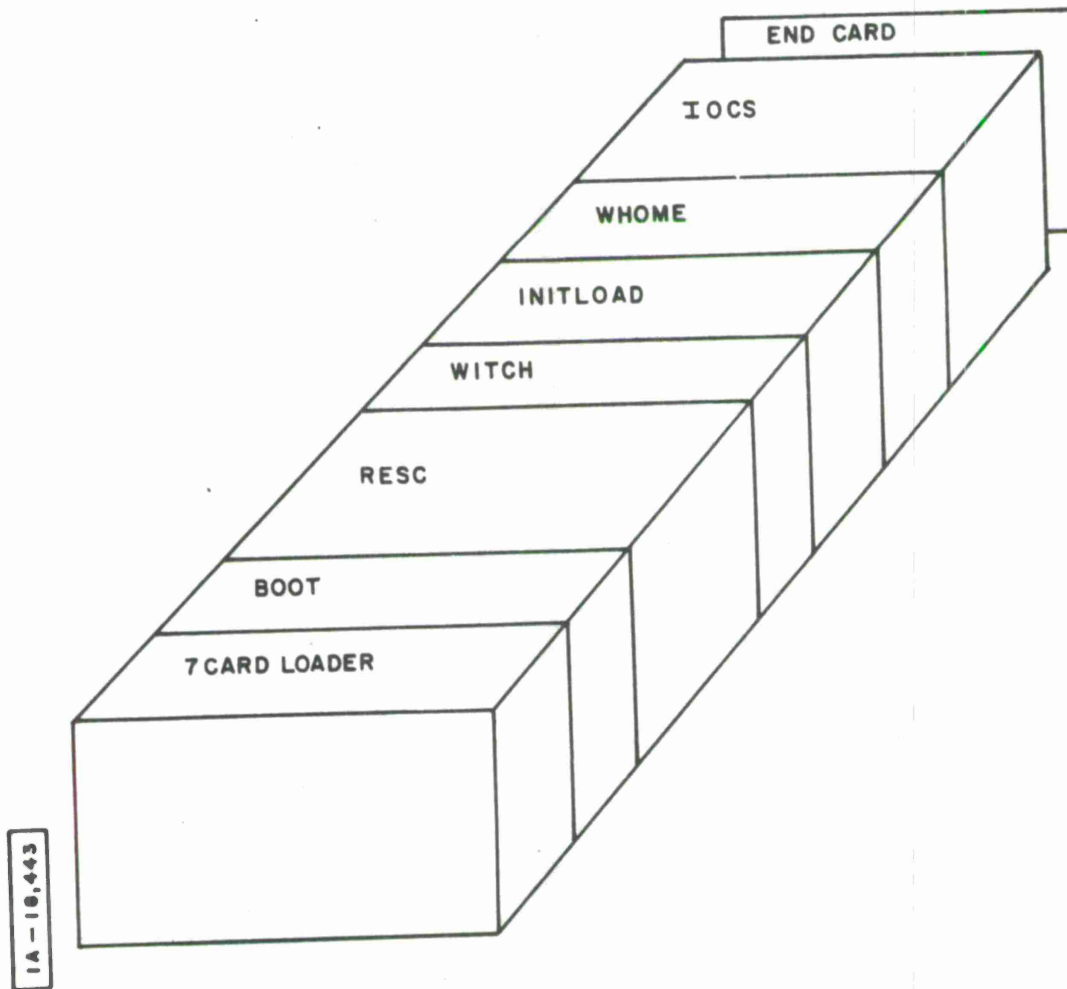
Figure 4. System Deck

13

Bootstrap Instruction for Loading from Cards

(a) Turn mode switch to DISPLAY; press START.
Type 00247. Blanks should be printed,
since core is empty.

(b) Turn mode switch to ALTER; press START.
Type L(1100257RR

(c) Turn mode switch to ADDRESS SET; press
START. Type 00247.

(d) Turn mode switch to RUN; then press
START.

The system is now prepared to learn what configuration it is to use. This is explained following the tape loading instructions.

Loading from Tape Onto Disk

Deck Setup for Loading from Tape
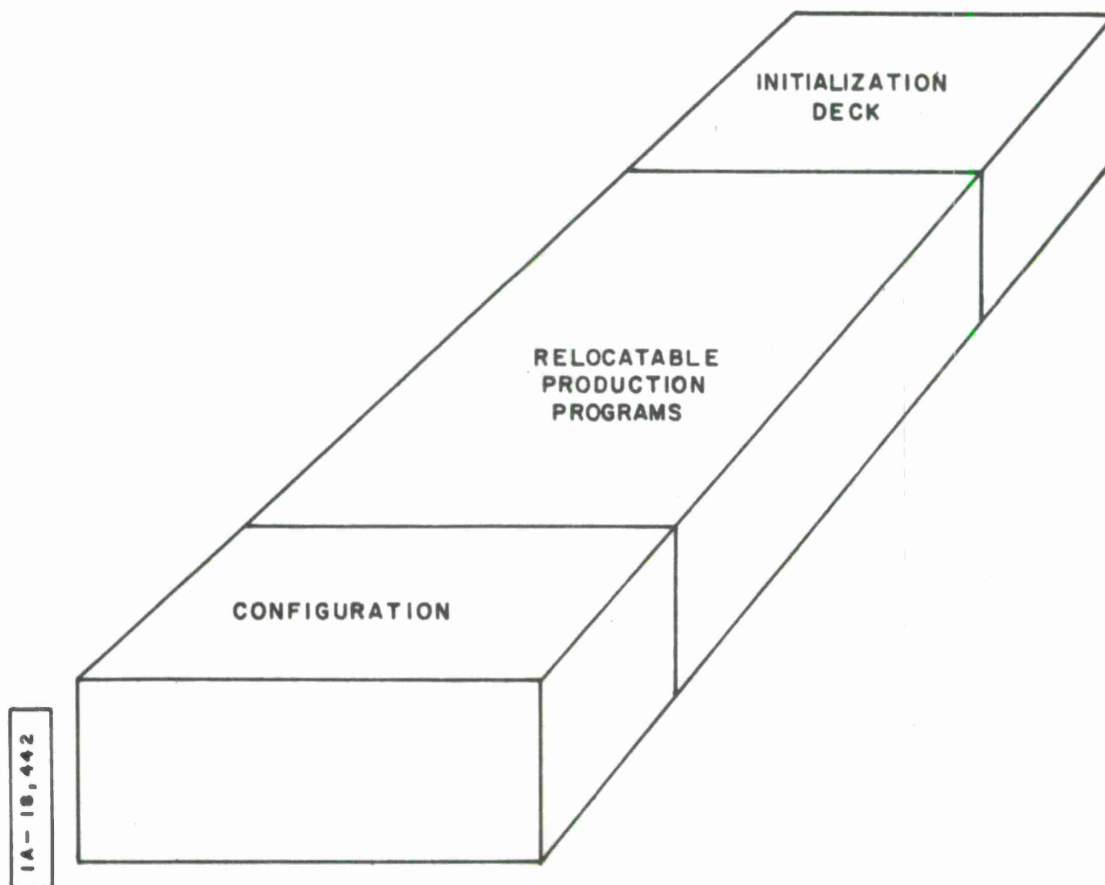
Note that even when loading from tape, some cards are still used: the configuration deck (optional) and the initialization deck (required). Although most of the relocatable production programs would normally be on tape, any which are not may be included between the configuration deck and the initialization deck.

Figure 5. Card Deck Used When Loading from Tape

15

Bootstrap Instruction for Loading from Tape*

    (a) Turn mode switch to DISPLAY; press START.
Type 00001. Blanks should be printed,
since core is empty.

    (b) Turn mode switch to ALTER; press START.
Type $\overset{v}{L}x_1Bx_200012R\overset{v}{N}$

where

        x_1 is the channel designation, either a
left parenthesis, (, for channel 1 or
a right parenthesis,), for channel 2.

        x_2 is the tape drive number which the
C-10 system tape is on.

    (c) Turn mode switch to RUN; press COMPUTER
RESET; press START.

---

* Note: The tape may be mounted on <u>any</u> drive.

Configuration Instructions

As soon as the initial loading instructions stated above have
been executed, a series of questions will be asked by the C-10 loader.
This is to aid the operator to state what apparatus he is using, what
channel each is attached to, etc. The answers are typed in or may be
read in part from the card reader (the optional configuration deck).

Following this question-answer routine, the system punches out
a bootstrap card which contains, in coded form, details about the
configuration. This bootstrap card is used at a later time when-
ever the operator wishes to use the system.

The following flowchart gives a general description of the
instructions for stating configuration changes. A detailed descrip-
tion follows. The instructions are considered "changes", since the
system assumed certain equipment is present; to state the true con-
figuration, changes (and additions or deletions) are made to this
assumed configuration.

Capital letters indicate actual typed messages, while small
letters indicate an action (by either the system or the operator).
The R's begin responses by the system; the I's begin statements which
are typed by the operator.
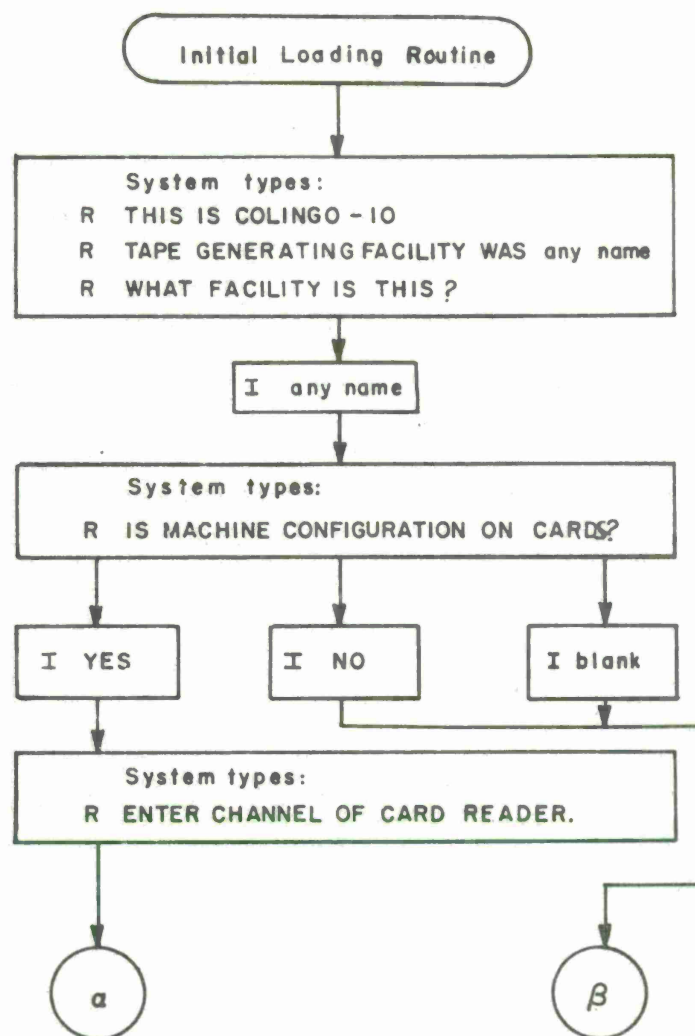
Figure 6. Configuration Changes
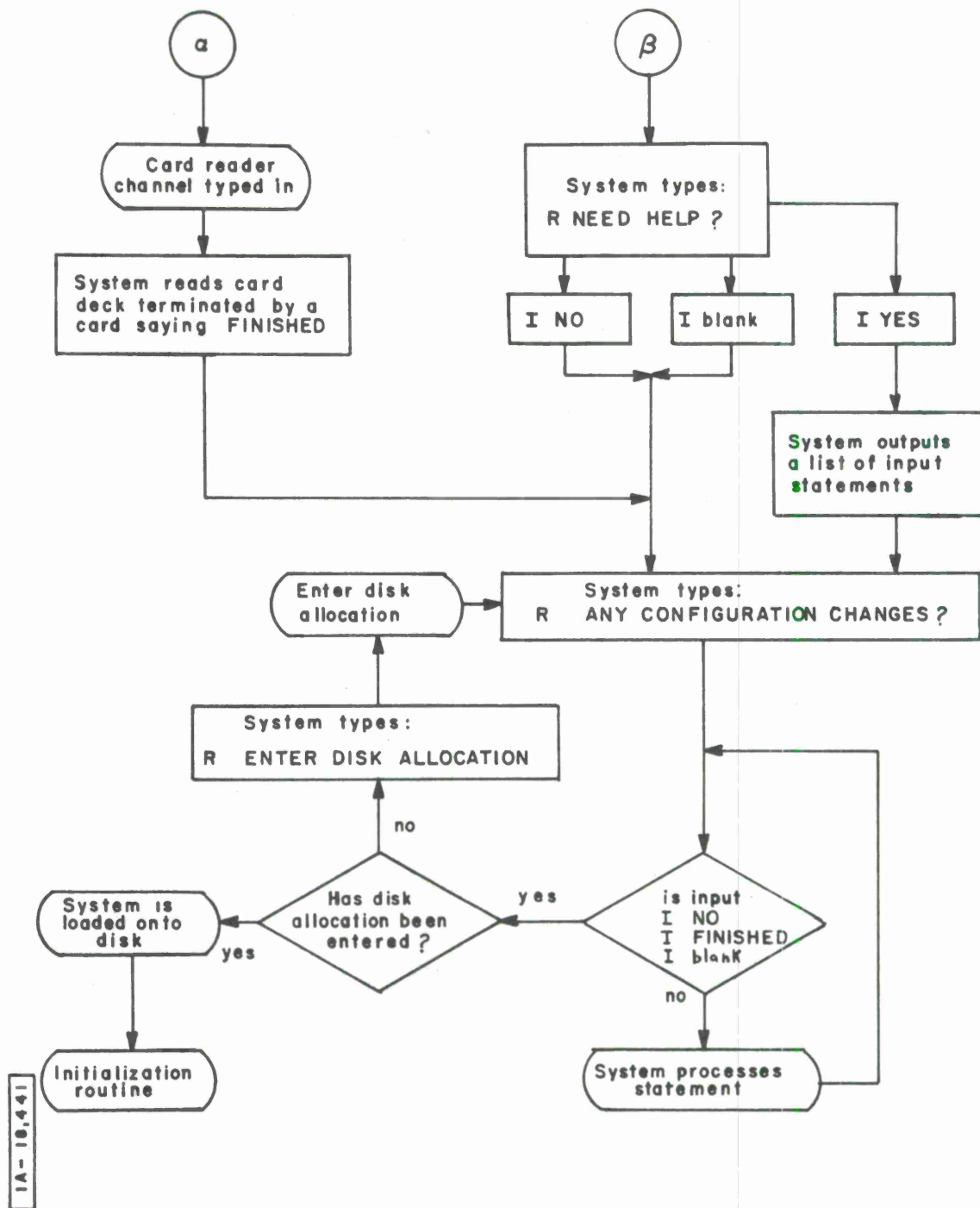
Figure 6(Concluded). Configuration Changes

Format for Stating Configuration Changes

A configuration change is made by typing an input statement which consists of a key word followed by some special parameters separated by commas. In a few cases, the input statement may consist of just a key word with no parameters. Thus input statements for stating configuration changes are of the form:

        key word,parameter 1,parameter 2,...

or

        key word

The key words must begin in Column 1. The following are the permissible key words:

            HELP
            CLEAR
            CLOCK
            PUNCH
            DELETE
            CONSOLE
            PRINTER
            DESCRIBE
            FINISHED
            NO. TAPES
            1301 DISK
            1302 DISK
            1311 DISK
            TAPE SELECT
            CARD READER
            AUTO START
            CORE SIZE

HELP, CLEAR, FINISHED, CLOCK and AUTO START are the only key words which may be used by themselves as input statements.

(a) HELP

This key word causes the system to type out
a list of key words and input statements.
This is to aid the operator in starting a
configuration change when he is unfamiliar
with the format for doing so.

(b) CLEAR

This key word clears the system of all know-
ledge of the current configuration. The
entire configuration must now be restated.

(c) FINISHED

This key word is used when all necessary
configuration changes have been stated.

(d) CLOCK

This key word announces the presence of a
clock.

(e) AUTO START

This key word is used to announce the existence
of the auto start feature on this machine.

The following describes the acceptable input statements
which need parameters applied to them. Note that in all cases the
first word is one of the key words listed above. The words follow-
ing are parameters (with the exception of DELETE,...which begins
with two key words) into which values must be substituted. When
the parameter "channel" appears, one of the integers, 1 or 2, must
be substituted to specify the channel under discussion. No spaces
may occur within an input statement.

If the operator is loading from a tape (or card deck) not made at that facility, he should ask to have the two channels described (a command exists for doing so; see below), and make the appropriate changes to the assumed configuration.

It is possible to state the actual configuration changes on cards, or the typewriter, or a combination of the two.

(f) DESCRIBE, Channel

This is a request for the system to type out a list of the equipment attached (as far as the system is aware) to the indicated channel. If the tape or card deck has ever been loaded before, a configuration has been stated, and this configuration is remembered by the system. For this reason it is a good idea to have both channels described whenever loading.

Example:

DESCRIBE,2

(g) CORE SIZE, Number

Only 40K, 60K, or 80K core sizes may be specified.

Example:

CORE SIZE,60K

(h) PUNCH, Channel, Device No.

This key word indicates a card punch. Device number is used when more than one punch is available. If there is only one punch, this parameter may be left blank.

Example:

PUNCH,2

(i)　NO. TAPES,Channel,Number

This key word is used to state the number of
tapes. If there are tapes on both channels,
two separate statements must be used.

Example:

NO. TAPES,1,5

(j)　PRINTER,Channel,Device No.

This key word refers to a line printer which
prints 132 characters per line.

Example:

PRINTER,1,1

(k)　CONSOLE,Channel,Device No.

This key word refers to the typewriter.

Example:

CONSOLE,2,1

(1)　CARD READER,Channel,Device No.

Example:

CARD READER,1,1

(m)　TAPE SELECT,Channel,Unit,Unit...

This key word denotes that tapes are attached
on the indicated channel, and that the drives
may be set to any of the unit select lines
stated.

Example:

TAPE SELECT,2,0,2,4,6,8

23

(n) 1301 DISK,Channel,Module,Access,
     Begin Track,End Track

     The disk module is represented by an integer
     from 0-9, the access by a 0 or 1. Begin track
     and end track are four-digit numbers indicating
     the track boundaries that the system is to
     occupy. Care should be taken so as not to
     reference disk space occupied by other non C-10
     users. A 1302 disk is also acceptable with the
     same type parameters.

(o) 1311 DISK,Channel,Begin Track,End Track

     There are no module and access parameters used
     to refer to a 1311 disk. However, in the begin
     track and the end track parameters, the 1311 disk
     addresses must be converted to 1301 form in
     accordance with C-10 disk mapping. See Appendix II.
     No disk configuration is assumed; every time the
     system is loaded the disk configuration must be
     stated.

(p) DELETE,Input Statement

     The keyword DELETE, followed by one of the input
     statements, will delete the particular
     apparatus (on the particular channel, etc.)
     mentioned in the input statement.

24

Example 1

```
=  ØØØØØ
D  ØØØØ1
D  bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
   bbbbbbbbbbbbbbbbbbbbbbbbbbbbb
A  Ĺ(B1ØØØ12RÑ
B  ØØØØ1
R  THIS IS COLINGO-1Ø
R  TAPE GENERATING FACILITY WAS MITRE
R  WHAT FACILITY IS THIS?
I  MITRE
R  IS MACHINE CONFIGURATION ON CARD?
I  YES
R  ENTER CHANNEL OF CARD READER
I  1
R  ANY CONFIGURATION CHANGES?
I  HELP
R         KEYWORDS
R
R  HELP
R  CLEAR
R  CLOCK
R  PUNCH
R  DELETE,
R  CONSOLE
R  PRINTER
R  DESCRIBE
R  FINISHED
R  NO. TAPES
R  13Ø1 DISK
R  13Ø2 DISK
R  1311 DISK
R  TAPE SELECT
R  CARD READER
R  AUTO START
R  CORE SIZE
R
R
```

Example 1 Continued

```
R        INPUT STATEMENT
R
R  CLEAR
R  CLOCK
R  AUTO START
R  DESCRIBE,CHANNEL
R  CORE SIZE,NUMBER
R  PUNCH,CHANNEL,DEVICE NO.
R  NO. TAPES,CHANNEL,NUMBER
R  PRINTER,CHANNEL,DEVICE NO.
R  CONSOLE,CHANNEL,DEVICE NO.
R  CARD READER,CHANNEL,DEVICE NO.
R  TAPE SELECT,CHANNEL,UNIT,UNIT,...
R  1311 DISK,CHANNEL,BEGIN TRACK,END TRACK
R  1301 DISK,CHANNEL,MODULE,ACCESS,BEGIN TRACK,END TRACK
R  1302 DISK,CHANNEL,MODULE,ACCESS,BEGIN TRACK,END TRACK
R
R  DELETE,    INPUT STATEMENT
R
R
R ANY CONFIGURATION CHANGES?
I 1311 DISK,2,0200,0800
I FINISHED
```

The first four lines are the initial loading routine. The state-
ment THIS IS COLINGO-10 indicates that the initial loading section was
successfully completed. The R's before each line denote response and
precede questions and statements by the system. The I's denote inquiry
and are typed by the person sitting at the typewriter (Inquiry Request
Button).

Lines 9 and 10 say that the system was loaded at The MITRE Corporation. A deck setup similar to that shown on the following page was used with the card reader on channel 1. The system then asked if there were any changes in the configuration. By typing HELP, (line 16), he causes the system to print out a list of key words and input statements. After this list was typed by the system, it asked if there were any configuration changes to be made. The operator typed in that there was a 1311 disk on channel 2 with beginning and end track numbers of 0200 and 0800 respectively. Since this was the last change to be made, he then typed in the key word FINISHED. At this point, the system normally punches out the bootstrap card; but, since the punch was not ready, the system asked the operator to ready it (READY PUNCH).

FINISHED

CARD READER, 1, 1

PRINTER, 1, 1

CONSOLE, 1, 1

1311 DISK, 2, 0360, 0800

CORE SIZE, 40K

AUTO START

CLEAR

IA— 10,444
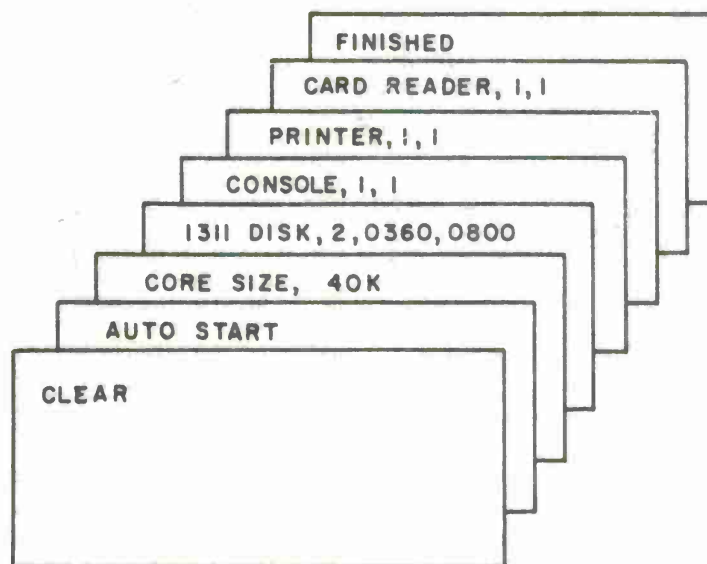
Figure 7. Example of Card Setup for Configuration Change

Example 2

```
  = 00000
  D 00001
  D bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
    bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
  A [(B100012RÑ
  B 00001
  R THIS IS COLINGO-10
  R TAPE GENERATING FACILITY WAS MITRE
  R WHAT FACILITY IS THIS?
  I MITRE
  R IS MACHINE CONFIGURATION ON CARD?
  I NO
  R NEED HELP?
  I
  R ANY CONFIGURATION CHANGES?
  I DESCRIBE,1
  R   CONSOLE,1,1
  R   PRINTER,1,1
  R   PUNCH,1,1
  R   CARD READER,1,1
  R   TAPE SELECT,1,0
  R   TAPE SELECT,1,1
  R   TAPE SELECT,1,2
  R   TAPE SELECT,1,3
  R   TAPE SELECT,1,4
  R   TAPE SELECT,1,5
  R   TAPE SELECT,1,6
  R   TAPE SELECT,1,7
  R   TAPE SELECT,1,8
  R   TAPE SELECT,1,9
  R   NO. TAPES,1,5
  I DELETE,TAPE SELECT,1,0,2,4,6,8
  I TAPE SECEL
  I TAPE SELECT,2,0,2,4,6,8
  I PUNCH,2,2
  I CARD READER,2,3
  I CARD READER,1,5
  I CONSOLE,1,3
  I PRINTER,1,4
  I PUNCH,1,2
  I DESCRIBE,1
  R   CONSOLE,1,1
  R   PRINTER,1,1
```

29

Example 2 (Continued)

```
R  PUNCH,1,1
R  CARD READER,1,1
R  TAPE SELECT,1,Ø
R  TAPE SELECT,1,1
R  TAPE SELECT,1,2
R  TAPE SELECT,1,3
R  TAPE SELECT,1,4
R  TAPE SELECT,1,5
R  TAPE SELECT,1,6
R  TAPE SELECT,1,7
R  TAPE SELECT,1,8
R  TAPE SELECT,1,9
R  CARD READER,1,5
R  CONSOLE,1,3
R  PRINTER,1,4
R  PUNCH,1,2
R  NO. TAPES,1,5
I  FINISHED
```

Example 3

```
=  ØØØØØ
D  ØØØØ1
D  bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
   bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
   bbbbbbbbbb
A  [(B1ØØØ12RÑ
B  ØØØØ1
R  THIS IS COLINGO-1Ø
R  TAPE GENERATING FACILITY WAS MITRE
R  WHAT FACILITY IS THIS?
I  MITRE
R  IS MACHINE CONFIGURATION ON CARD?
I  YES
R  ENTER CHANNEL OF CARD READER
I  1
R  ANY CONFIGURATION CHANGES?
I  NO
```

30

Example 4

```
 = ØØØØØ
 D ØØØØ1
 D bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
   bbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
 A [(B1ØØØ12RM
 B ØØØØ1
 R THIS IS COLINGO-1Ø
 R TAPE GENERATING FACILITY WAS MITRE
 R WHAT FACILITY IS THIS?
 I MITRE
 R IS MACHINE CONFIGURATION ON CARD?
 I YES
 R ENTER CHANNEL OF CARD READER
 I 1
 R ANY CONFIGURATION CHANGES?
 I DELETE,1311 DISK,2
 I 1311 DISK,2,Ø36Ø,Ø8ØØ
 I DESCRIBE,1
 R   CONSOLE,1,1
 R   PRINTER,1,1
 R   PUNCH,1,1
 R   CARD READER,1,1
 R   TAPE SELECT,1,Ø
 R   TAPE SELECT,1,1
 R   TAPE SELECT,1,2
 R   TAPE SELECT,1,3
 R   TAPE SELECT,1,4
 R   TAPE SELECT,1,5
 R   TAPE SELECT,1,6
 R   TAPE SELECT,1,7
 R   TAPE SELECT,1,8
 R   TAPE SELECT,1,9
 R   NO. TAPES,1,5
 I FINISHED
 R READY PUNCH
```

31

Example 5

```
=  00000
D  00001
D  bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbbbbbb
A  [CP100012RM
P  00001
R  THIS IS COLINGO-10
R  TAPE GENERATING FACILITY WAS MITRE
R  WHAT FACILITY IS THIS?
I  IBM
R  IS MACHINE CONFIGURATION ON CARD?
I  NO
R  NEED HELP?
I
R  ANY CONFIGURATION CHANGES?
I  DESCRIBE,1
R   CONSOLE,1,1
R   PRINTER,1,1
R   PUNCH,1,1
R   CARD READER,1,1
R   TAPE SELECT,1,0
R   TAPE SELECT,1,1
R   TAPE SELECT,1,2
R   TAPE SELECT,1,3
R   TAPE SELECT,1,4
R   TAPE SELECT,1,5
R   TAPE SELECT,1,6
R   TAPE SELECT,1,7
R   TAPE SELECT,1,8
R   TAPE SELECT,1,9
R   NO. TAPES,1,5
I  DESCRIBE,2
R   NO. TAPES,2,!
I  1311 DISK,2,0200,1640
I  DESCRIBE,2
R   NO. TAPES,2,!
P   1311 DISK
I  FINISHED
```

32

Example 6

```
=  00000
D  00001
D  bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
   bbbbbbbbbbbbbbbbbbbbbbbbb
A  [(B100012RN
R  00001
R  THIS IS COLINGO-10
R  TAPE GENERATING FACILITY WAS MITRE
R  WHAT FACILITY IS THIS?
I  MITRE
R  IS MACHINE CONFIGURATION ON CARD?
I  NO
R  NEED HELP?
I  YES
R          KEYWORDS
R
R   HELP
R   CLEAR
R   CLOCK
R   PUNCH
R   DELETE,
R   CONSOLE
R   PRINTER
R   DESCRIBE
R   FINISHED
R   NO. TAPES
R   1301 DISK
R   1302 DISK
R   1311 DISK
R   TAPE SELECT
R   CARD READER
R   AUTO START
R   CORE SIZE
R
R
R          INPUT STATEMENT
R
R   CLEAR
R   CLOCK
R   AUTO START
R   DESCRIBE,CHANNEL
R   CORE SIZE,NUMBER
R   PUNCH,CHANNEL,DEVICE NO.
R   NO. TAPES,CHANNEL,NUMBER
R   PRINTER,CHANNEL,DEVICE NO.
R   CONSOLE,CHANNEL,DEVICE NO.
R   CARD READER,CHANNEL,DEVICE NO.
R   TAPE SELECT,CHANNEL,UNIT,UNIT,...
R   1311 DISK,CHANNEL,BEGIN TRACK,END TRACK
R   1301 DISK,CHANNEL,MODULE,ACCESS,BEGIN TRACK,END TRACK
R   1302 DISK,CHANNEL,MODULE,ACCESS,BEGIN TRACK,END TRACK
R
R   DELETE,    INPUT STATEMENT
R
```

Example 6 (Continued)

```
R
R ANY CONFIGURATION CHANGES?
I DESCRIBE,1
R   CONSOLE,1,1
R   PRINTER,1,1
R   PUNCH,1,1
R   CARD READER,1,1
R   TAPE SELECT,1,0
R   TAPE SELECT,1,1
R   TAPE SELECT,1,2
R   TAPE SELECT,1,3
R   TAPE SELECT,1,4
R   TAPE SELECT,1,5
R   TAPE SELECT,1,6
R   TAPE SELECT,1,7
R   TAPE SELECT,1,8
R   TAPE SELECT,1,9
R   NO. TAPES,1,5
I DESCRIBV
I DESCRIBE,2
R   NO. TAPES,2,!
I DELETE,NO. TAPES,2
I DESCROBE,2
I DESCI
I DESCRIBE,2
I 1311 DISK,2,0200,1640
I DESCRIBE,2
R   1311 DISK
I FINISHED
```

Initialization

After the system has been loaded onto disk, and before
it can be run, the initialization process must be used to clear
the portions of the disk that are needed for data storage. This
process also initializes such functions as STEP, the Master Index,
block management, disk allocation, etc. The initialization deck
has already been put in the card reader behind the RELOCATABLE
PRODUCTION PROGRAMS deck (see Figures 2, 3 and 5). It is only
necessary to type 154, the number of the initialization subroutine.
However, initialization should not begin until an ENTER STARTING
SUBROUTINE message appears.

        R   ENTER STARTING SUBROUTINE
        I   154
        R   ENTER STARTING SUBROUTINE

Re-initialization may also be done at any later time where it is
desired to "wipe the disk clean" of file structures and procedures.
Caution should be taken in this operation, however, so as not to
lose wanted procedures. Re-initialization is accomplished as
described for initialization above.

OPERATING PROCEDURES

The operating procedures essentially explain how to operate C-10 from disk once it has been loaded from tape or card onto disk.*

Deck Setup for Loading from Disk

Single bootstrap card (punched by system at tape or card load time) is needed.

Instructions for Loading from Disk

(a) Ready card reader with single bootstrap card.

(b) Turn mode switch to DISPLAY; press START; type 00001

(c) Turn mode switch to ALTER; press START; type

L(1100012RR

(d) Turn mode switch to RUN; press COMPUTER RESET; press START.

(e) ENTER STARTING SUBROUTINE is printed out. At this time, the number of any C-10 subroutine to be executed may be entered. Normally, 237 is entered to call the EXEC and begin normal operation. Examples of other uses include 154 for initialization (see above) and 295 to generate a new C-10 system tape.

---

*If the system has just been loaded from tape or cards, these steps are not necessary, and ENTER STARTING SUBROUTINE should appear automatically after the Initialization process.

Example 8

This is an example of loading from tape onto disk and
then from disk into memory.

```
D    ØØØØ1
D    bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
     v        v
A    L(B000012RN
B    ØØØØ1

R    THIS IS C-10
R    WHAT FACILITY IS THIS?
I    IBM
R    IS MACHINE CONFIGURATION ON CARD?
I    NO
R    NEED HELP?
I    NO
R    ANY CONFIGURATION CHANGES?
I    1311 DISK,2,Ø2ØØ,164Ø
I    DESCRIBE,2
R    NO.TAPES,2,1
R    1311 DISK
I    FINISHED
R    READY PUNCH
R    ENTER STARTING SUBROUTINE
I    154
R    ENTER STARTING SUBROUTINE
I    237
R    13.53 Ø5JAN66
R    PROFILE READY
```

37

Example 9

This example illustrates loading directly from disk (using the bootstrap card). Starting with line 9, two STEP programs are also shown.

```
D    ØØØØ1
D    bbᴸbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
     v        v
A    L(1100012RR
B    ØØØØ1
R    ENTER STARTING SUBROUTINE
I    237
R    14.58   1ØJAN66
R    PROFILE READY
I    STEP(PRINT(MIRE 'TOCS' 'BA))
R    ((PROG() (SETG 11 NIL) (SUPERCLA NIL) (PRINT
      MIRE 'TOCS 'BA)) (FILEOUT) (RETURN NIL) ))
R    NIL
R    PROFILE READY
I    STEP(SORTR)
R    ((PROG() (SETG 11 NIL) (SUPERCLA NIL) (SORTR)
       (FILEOUT) (RETURN NIL) ))
R
R    NIL
R    PROFILE READY
```

Instructions for Using C-10 Languages

In the operating procedures below, we make the distinction between hard procedures and soft procedures. A hard procedure is a procedure which has been added to the system ultimately in the form of relocatable 1410 machine code. A soft procedure is a procedure which has been added to the system ultimately in STEP interpretable form. Procedures which have not been added to the system are neither hard nor soft.

Procedures may be written in three languages: PROFILE, STEP, or AUTOCODER. Procedures in PROFILE or STEP may contain terses and actors.

Soft procedures are easy to enter into the system. Hard procedures are not as easy to enter into the system because it is necessary for the procedure to be assigned a number by a member of the C-10 maintainence group, it must be processed off-line by the AUTOCODER assembler, and the "BEGIN macro" must be changed prior to its assembly. Soft procedures may appeal to any procedure. Hard procedures may appeal only to hard procedures.

Procedures written in AUTOCODER are necessarily hard procedures. Procedures written in PROFILE or STEP may become either hard procedures or soft procedures.

Two versions of all hard procedures may be maintained by the system: an "experimental" version and a "production" version. Only one version of soft procedures is maintained by the system. The C-10 system may be operated in two modes: experimental or production. In experimental mode all existing experimental versions of hard procedures are used in conjunction with the production versions of those procedures for which there are no experimental versions. In production mode, only production versions of procedures are used.

The charts below provide instructions for entering and
executing messages, for adding procedures to the system, for converting
soft procedures into hard procedures, for changing the mode of system
operation from production to experimental, for aborting messages, and
for changing the "standard" input and output devices. The order of
the charts is as follows:

I.   To enter and execute PROFILE statements which contain no
        terses or actors.

II.   To enter and execute PROFILE statements which contain pre-
        processor terses or actors.

III.   To enter and execute PROFILE statements which contain co-
        processor terses or actors.

IV.   To enter and execute PROFILE statements which contain both
        pre-processor and co-processor terses and actors.

V.   To enter and execute STEP expressions which contain no terses
        and actors.

VI.   To enter and execute STEP expressions which contain pre-
        processor terses or actors.

VII.   To add a PROFILE procedure to the system as a soft procedure.

VIII.   To add a STEP procedure to the system as a soft procedure.

IX.   To convert a soft procedure into a hard procedure.

X.   To add an AUTOCODER procedure to the system.

XI.   To load AUTOCODER object decks as Experimental hard procedures.

XII.   To load AUTOCODER object decks as Production hard procedures.

XIII.   To change the mode of system operation from Production to
        Experimental.

XIV.   To abort the execution of a message.

XV.   To change the standard input and output devices.

XVI.   To compile PROFILE statements directly into AUTOCODER.

XVII.   To compile STEP expressions directly into AUTOCODER.

Additional information may be found in the next Section, Executive/Editor.

CHART I: To Enter and Execute PROFILE statements which contain no
           terses or actors.

Step	Action	Remarks
1	Ready card reader with single bootstrap card.	Steps 1 through 5 "load" C-10 from disk. If "select subroutine" is in core, steps 1 through 5 may be avoided by branching to location 39000. and typing 30 when the system responds "SELECT SUBROUTINE."
2	Turn mode switch to DISPLAY; press START; type 00001.	
3	*Turn mode switch to ALTER; press START; type L(1100012RR.	
4	Turn mode switch to ADDRESS SET; press START; type 00001.	
5	Turn mode switch to RUN; press START.	System responds "ENTER STARTING SUBROUTINE."
6	Type 237.	System responds "PROFILE READY."
7	Type PROFILE statements followed by a " Δ."	The PROFILE statements will be automatically translated into STEP language and executed as a soft procedure by the STEP interpreter. The system will respond "PROFILE READY" after execution, whereupon new statements may be entered.

---

*The funny looking character above the L and above the R is a "word
mark" and is typed by depressing the WORD MARK key before typing
L or R.

CHART II: To Enter and Execute PROFILE statements which contain
pre-processor terses or actors.

Step	Action	Remarks
1	Ready card reader with single bootstrap card.	Steps 1 through 5 "load" C-10 from disk. If "select subroutine" is in core, steps 1 through 5 may be avoided by branching to location 39000, and typing 30 when the system responds "SELECT SUBROUTINE."
2	Turn mode switch to DISPLAY; press START; type 00001.	
3	Turn mode switch to ALTER; press START; type L(1100012RR.	
4	Turn mode switch to ADDRESS SET; press START; type 00001.	
5	Turn mode switch to RUN; press START.	System responds "ENTER STARTING SUBROUTINE."
6	Type 237.	System responds "PROFILE READY."
7	Turn mode switch to DISPLAY; press START; type 37651.	
8	Turn mode switch to ALTER; press START; type J.	
9	Turn mode switch to RUN; press Start.	
10	Type PROFILE statements followed by a "Δ."	The PROFILE statements will be automatically processed by TAP, translated into STEP language and executed as a soft procedure by the STEP interpreter. The system will respond "TAP READY" after execution whereupon new statements may be entered.

43

CHART III: To Enter and Execute PROFILE statements which contain
co-processor terses or actors.

Step	Action	Remarks
1	Ready card reader with single bootstrap card.	Steps 1 through 5 "load" C-10 from disk. If "select subroutine" is in core, steps 1 through 5 may be avoided by branching to location 39ØØØ, and typing 3Ø when the system responds "SELECT SUBROUTINE."
2	Turn mode switch to DISPLAY; press START; type ØØØØ1.	
3	Turn mode switch to ALTER; press START; type L(I1ØØØ12RR.	
4	Turn mode switch to ADDRESS SET; press START; type ØØØØ1.	
5	Turn mode switch to RUN; press START.	System responds "ENTER STARTING SUBROUTINE."
6	Type 237.	System responds "PROFILE READY."
7	Turn mode switch to DISPLAY; press START; type 3765Ø.	
8	Turn mode switch to ALTER; press START; type J.	
9	Turn mode switch to RUN; press START.	
10	Type PROFILE statements followed by a " Δ."	The PROFILE statements will be automatically processed by TAP, translated into STEP language and executed as a soft procedure by the STEP interpreter. The system will respond " TAP READY" after execution, whereupon new statements may be entered.

CHART IV: To Enter and Execute PROFILE statements which contain both
pre-processor and co-processor terses and actors.

Step	Action	Remarks
1	Ready card reader with single bootstrap card.	Steps 1 through 5 "load" C-10 from disk. If "select subroutine" is in core, steps 1 through 5 may be avoided by branching to location 39000, and typing 30 when the system responds "SELECT SUBROUTINE."
2	Turn mode switch to DISPLAY; press START; type 00001.	
3	Turn mode swtich to ALTER; press START; type L(1100012RR.	
4	Turn mode switch to ADDRESS SET; press START; type 00001.	
5	Turn mode switch to RUN; press START.	System responds "ENTER STARTING SUBROUTINE."
6	Type 237.	System responds "PROFILE READY."
7	Turn mode switch to DISPLAY; press START; type 37650.	
8	Turn switch to ALTER; press START; type JJ.	
9	Turn mode switch to RUN; press START.	
10	Type PROFILE statements followed by a "△."	The PROFILE statements will be automatically processed by TAP, translated into STEP language and executed as a soft procedure by the STEP interpreter. The system will respond "TAP READY" after execution, whereupon new statements may be entered.

CHART V: To Enter and Execute STEP expressions which contain no
terses or actors.

Step	Action	Remarks
1	Ready card reader with single bootstrap card.	Steps 1 through 5 "load" C-10 from disk. If "select subroutine" is in core, steps 1 through 5 may be avoided by branching to location 39000, and typing 30 when the system responds "SELECT SUBROUTINE."
2	Turn mode switch to DISPLAY, press START; type 00001.	
3	Turn mode switch to ALTER; press START; type L(1100012RR.	
4	Turn mode switch to ADDRESS SET; press START; type 00001.	
5	Turn mode switch to RUN; press START.	System responds "ENTER STARTING SUBROUTINE."
6	Type 237.	System responds "PROFILE READY."
7	Turn mode switch to DISPLAY; press START; type 37652.	
8	Turn mode switch to ALTER; type 2.	
9	Turn mode switch to RUN; press START.	
10	Type STEP expressions followed by a "Δ."	The STEP expressions will be evaluated by the STEP interpreter. The system will respond "STEP READY" after execution, whereupon new statements may be entered.

CHART VI: To Enter and Execute STEP expressions which contain pre-
           processor terses or actors.

Step	Action	Remark
1	Ready card reader with single bootstrap card.	Steps 1 through 5 "load" C-10 from disk. If "select subroutine" is in core, steps 1 through 5 may be avoided by branching to location 39000, and typing 30 when the system responds "SELECT ROUTINE."
2	Turn mode switch to DISPLAY; press START; type 00001.	
3	Turn mode switch to ALTER; press START; type L(11000012RR.	
4	Turn mode switch to ADDRESS SET; press START; type 00001.	
5	Turn mode switch to RUN; press START.	System responds "ENTER STARTING SUBROUTINE."
6	Type 237.	System responds "PROFILE READY."
7	Turn mode switch to DISPLAY; press START; type 37651.	
8	Turn mode switch to ALTER; type J2.	
9	Turn mode switch to RUN; press START.	
10	Type STEP expressions followed by a "Δ."	The STEP expressions will be evaluated by the STEP interpreter. The system will respond "STEP READY" after execution, whereupon new statements may be entered.

CHART VII: To Add a PROFILE procedure to the system as a soft
            procedure.

Step	Action	Remarks
1	Enclose the procedure in a procedure declaration (see ESD-TR-66-653, Vol. I, Section III).	A soft PROFILE procedure is defined when a PROFILE procedure declaration is translated into STEP language and executed.
2	Enter and execute the procedure declaration	See CHART I.

CHART VIII: To Add a STEP procedure to the system as a soft procedure.

Step	Action	Remarks
1	Enclose the procedure in a DEFINE form (see Section IV, page 99.)	Soft procedures are defined when the STEP interpreter processes an appeal to the procedure DEFINE (See Section IV.) Hence, this operation may actually occur in the middle of the execution of any STEP procedure or expression or, for that matter, during the execution of any PROFILE procedure.
2	Execute the DEFINE form.	See CHART V.

CHART IX: To Convert a Soft Procedure into a Hard Procedure.

Step	Action	Remarks
1	Ready card reader with single bootstrap card.	Steps 1 through 5 "load" C-10 from disk. If "select subroutine" is in core, steps 1 through 5 may be avoided by branching to location 39000, and typing 30 when the system responds "SELECT SUBROUTINE."
2	Turn mode switch to DISPLAY; press START; type 00001.	
3	Turn mode switch to ALTER; press START; type L(11000012RR.	
4	Turn mode switch to ADDRESS SET; press START; type 00001.	
5	Turn mode switch to RUN; press START.	System responds "ENTER STARTING SUBROUTINE."
6	Type 237.	System responds "PROFILE READY."
7	Type STEP (XLSTEP QUOTE f_1 QUOTE f_2)	f_1 is the name of the existing soft procedure and f_2 is the name of the future hard procedure. The system will punch out an AUTOCODER deck and respond "PROFILE READY" when compilation is complete.
8	Add the AUTOCODER procedure to the system.	See CHART X.

CHART X: To Add an AUTOCODER Procedure to the System

Step	Action	Remarks
1	Get a number.	Each subroutine which is to be added eventually to the C-10 system must be assigned a number in the subroutine directory. Routine numbers are assigned by a member of the C-10 maintainence group.
2	Change BEGIN macro.	An addition must be made to the BEGIN macro. This consists of equating the subroutine name with the number assigned to it in the subroutine directory. This addition is usually made by the member of the C-10 maintainence group who allocates subroutine directory numbers.
3	Check deck setup.	The symbolic deck of the AUTOCODER procedure should be structured as indicated below:

```
   6        16           21
MON$$     JOB          Programmer, project,
                       dept, room
MON$$     EXEQ         AUTOCODER,,,NOFLG
          TITLE        Subroutine-name
          HEADR        (for the top of each page)
          PST          (if cross reference
                       listing desired)
          BEGIN        subroutine-name,
                       number of arguments,
                       number of words of
                       temporary storage, type
        (AUTOCODER statements)
          FIN
          END
```

Step	Action	Remarks
4	Submit deck for assembly.	A special version of PR-155, containing all C-10 macros and labeled "PALERMO C-10" is kept at the 1410 console in Bedford. The deck described above may be submitted with a 1410 request card on which is stated "please use PR-155 PALERMO C-10."
		Since the BEGIN macro expansion generates many pages of system definitions at the beginning of every assembly listing, an option to suppress this printout has been made available. If the option is specified PR-155 will generate its output listing on a magnetic tape rather than the printer. SELECT SUBROUTINE may then be used to print the tape, skipping the system definitions included in the BEGIN macro. If macro suppression is desired, the words "macro suppression" should be added to the request card.
		The result of an AUTOCODER assembly is an "object deck", punched on cards.
5	Load object deck as either an "Experimental" or a "Production" deck.	See Chart XI or Chart XII.

CHART XI: To Load AUTOCODER Object decks as Experimental Hard Procedures

Step	Action	Remarks
1	Check deck setup.	More than one object deck may be loaded at a time. The object decks should be stacked together and preceded by a card punched with the words "RESET EXPERIMENTAL." The word RESET must begin in Column 1.

RESET EXPERIMENTAL

Step	Action	Remarks
2	Ready card reader with single bootstrap card.	Steps 2 through 6 "load" C-10 from disk. If "select subroutine" is in core, Steps 2 through 6 may be avoided by branching to location 39000, and typing 30 when the system responds "SELECT SUBROUTINE."
3	Turn mode switch to DISPLAY; press START; type 00001.	
4	Turn mode switch to ALTER; press START; type ᵞL(1100012Rᵞ.	
5	Turn mode switch to ADDRESS SET; press START; type 00001.	
6	Turn mode switch to RUN; press START.	System responds "ENTER STARTING SUB-ROUTINE."
7	Ready card reader with deck.	
8	Type 237.	System will respond "PROFILE READY." The object decks have been loaded at this time. PROFILE statements may be entered.

CHART XII: To Load AUTOCODER Object decks as Production Hard Procedures

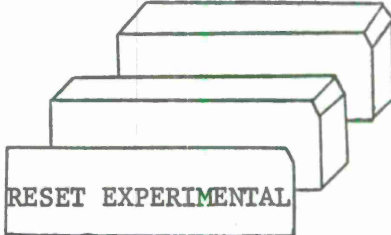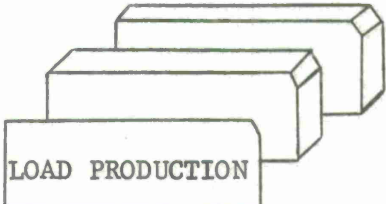| Step	Action	Remarks
1	Check deck setup.	More than one object deck may be loaded at a time. The object decks should be stacked together and preceded by a card punched with the words "LOAD PRODUCTION." The word LOAD must begin in Column 1.

LOAD PRODUCTION

Step	Action	Remarks
2	Ready card reader with single bootstrap card.	Steps 2 through 6 "load" C-10 from disk. If "select subroutine" is in core, Steps 2 through 6 may be avoided by branching to location 39000, and typing 30 when the system responds "SELECT SUBROUTINE."
3	Turn mode switch to DISPLAY; press START; type 00001.	
4	Turn mode switch to ALTER; press START; type L(1100012RR.	
5	Turn mode switch to ADDRESS SET; press START; type 00001.	
6	Turn mode switch to RUN; press START.	System responds "ENTER STARTING SUB-ROUTINE."
7	Ready card reader with deck.	System will respond "PROFILE READY." The object decks have been loaded at this time. PROFILE statements may be entered.
8	Type 237.	

CHART XIII: To change the mode of system operation from Production to
              Experimental

Step	Action	Remarks
1	Ready card reader with single bootstrap card.	Steps 1 through 5 "load" C-10 from disk. If "select subroutine" is in core, Steps 1 through 5 may be avoided by branching to location 39000, and typing 30 when the system responds "SELECT SUBROUTINE."
2	Turn mode switch to DISPLAY; press START; Type 00001.	
3	Turn mode switch to ALTER; press START; type L(11000012RR.	
4	Turn mode switch to ADDRESS SET; press START; type 00001.	
5	Turn mode switch to RUN; press START.	
6	Ready card reader with "USE EXPERIMENTAL" card.	"USE EXPERIMENTAL" should be punched in columns 1 through 16.
7	Type 237.	System will respond "PROFILE READY." PROFILE statements may be entered.

55

CHART XIV: To abort the execution of a message

Step	Action	Remarks
1	Press INQUIRY	System should respond "TYPE GO OR ABORT."
2	Type ABORT if execution is to be terminated; type GO if execution is to be continued.	

CHART XV: To change the "standard" input and output devices

<u>STEP</u>   <u>Action</u>

  1   The system should
          be set up to accept
          either STEP or PROFILE
          messages.

  2   If the system is accept-   Acceptable input devices
          ing STEP messages, type   are CARDS or CONSOLE.
          (CHIPS I.D. O.D.) Δ   Acceptable output devices
          Where I.D. is an accept-   are CARDS, CONSOLE or
          able input device and   PRINTER.
          O.D. is an acceptable
          output device.

  3   If the system is accepting   Acceptable input devices are
          PROFILE messages, type   CARDS or CONSOLE. Acceptable
          STEP (CHIPS I.D. O.D.)Δ   output devices are CARDS,
          Where I.D. is an   CONSOLE, or PRINTER
          acceptable input device
          and O.D. is an accept-
          able output device.

CHART XVI: To compile PROFILE statements directly into AUTOCODER

Step	Action	Remarks
1	Follow Chart I, II, III, or IV to set up system for PROFILE statements; but do not enter statements.	Use Chart I for no terses or actors, Chart II for pre-processor terses or actors, Chart III for co-processor terses or actors, Chart IV for both pre-processor and co-processor terses and actors.
2	Turn mode switch to DISPLAY; press START; type 37654.	
3	Turn mode switch to ALTER; press START; type 52.	
4	Turn mode switch to RUN; press START.	
5	Type PROFILE statements followed by a "Δ."	The PROFILE statements will be automatically translated into AUTO-CODER and a deck will be punched. No execution will take place. The system will finally respond "PROFILE READY," or "TAP READY."

CHART XVII: To compile STEP expressions directly into AUTOCODER

Step	Action	Remarks
1	Follow Chart V, or Chart VI to set up system for STEP expressions, but do not enter expressions.	Use Chart VI for terses and actors, Chart V otherwise.
2	Turn mode switch to DISPLAY; press START; type 37654.	
3	Turn mode switch to ALTER; press START; type 52.	
4	Turn mode switch to RUN; press START.	
5	Type STEP expressions followed by a "Δ."	The STEP expressions will be automatically translated into AUTOCODER and a deck will be punched. No execution will take place. The system will finally respond "STEP READY," or "TAP READY."

SECTION III

INTRODUCTION TO TERSES AND ACTORS

INTRODUCTION

TAP is a facility provided by C-10 for performing string operations on an input message. TAP processes its input message from beginning to end, performing its operations when it encounters key words in the message. TAP receives its input message from the Editor, which has broken it down into a sequence of atoms. For example, TAP would receive the message

FIND AIRFIELD, 1800, 75, BOSTON, 50;

as the sequence of atoms

FIND

AIRFIELD

,

1800

,

75

,

BOSTON

,

50

;

As far as TAP is concerned, there are three types of atoms in the input message: atoms of type terse, atoms of type actor, and atoms which are neither type terse nor actor. TAP knows which atoms are the names of terses and actors because it has a list of all the terses and actors in the system.

Associated with each terse name in TAP's list of terses is a "skeleton." Associated with each actor name in TAP's list of actors is a STEP expression.

A skeleton is a sequence of atoms, but some of the atoms in a skeleton are "dummy" atoms. When TAP encounters a terse in the input message, it removes it from the message and replaces it with the skeleton it finds associated with the name of the terse on its list of terses. Inside the skeleton TAP replaces the dummy atoms with other atoms from the input message, according to a set of rules that will be explained below.

For example, suppose that FIND is a terse, and is associated by TAP with the skeleton

```
PRINT SUBSET AIRFIELD
      IF ANY [RUNWAY (LENGTH) GE X
              AND RUNWAY (WIDTH) GE Y]
      AND DISTANCE ('Z'; AIRFIELD (LOCATION)) < XX
      (NAME, LOCATION, RUNWAY (LENGTH, WIDTH));
```

where X, Y, Z and XX are dummy atoms. Then when TAP processes the input message above, the result is the message below:

```
PRINT SUBSET AIRFIELD
      IF ANY [RUNWAY (LENGTH) GE 18ØØ
              AND RUNWAY (WIDTH) GE 75]
      AND DISTANCE ('BOSTON'; AIRFIELD (LOCATION)) < 5Ø
      (NAME, LOCATION, RUNWAY (LENGTH, WIDTH));
```

After TAP has substituted the skeleton associated with a terse for an occurrence of the name of the terse and its arguments in an input message, TAP resumes the processing of the message beginning with the first atom of the skeleton.

A STEP expression is the basic element of STEP language (See Section IV). In particular, a STEP expression can be the name of a procedure. When TAP encounters an actor in the input message, it evaluates the associated STEP expression. In the case where the STEP-expression is a procedure name, the procedure is executed.

61

Any procedures associated with an actor through the STEP expression may do anything at all, including changing the internal workings of TAP.

To summarize, we describe the TAP mechanism in slightly more detail. When TAP is asked to process an input message, it places it in a push-down stack called ASPS. An input message is a list of atoms. During the operation of TAP, ASPS contains a list of such lists of atoms. The list on the "bottom" is the input message. All other lists are either the result of a terse expansion (skeletons with the dummy atoms replaced by arguments) or the result of an action taken by an actor. TAP fetches atoms one at a time by appealing to the function GETATOM. GETATOM takes the first atom off the "top" list of atoms and places it in a global location called CURATOM. It then removes the atom from the list of atoms from whence it came. When the list becomes empty it is removed from ASPS. When another list of atoms is placed on ASPS by the expansion of a terse, or a special action performed by an actor, it is placed at the top of ASPS. That is, the next atom fetched will be the first atom of the new list.

The output of TAP is a list of atoms. TAP prepares the output list of atoms by repetitively calling the function GETAPATOM. GETAPATOM checks the global location CURATOM to see if it contains the name of a terse or actor. If it does not, the content of CURATOM is moved to the global location CURTAPATOM, an appeal is made to the function GETATOM to refill CURATOM, and GETAPATOM returns with the value of CURTAPATOM. If CURATOM is the name of a terse or actor, either a function called SETUPT is asked to expand the terse, or a function called EVAL is asked to evaluate the STEP expression associated with the actor. Upon the completion of this operation, the function GETATOM is asked to refill CURATOM, and the process is repeated (CURATOM is checked to see if it contains the name of a terse or actor, etc.).

TERSE DEFINITIONS

Terses can be defined by input messages to TAP in the following format:

$$DT \; tn \; (as_1; \ldots; as_n), \; (s), \; ewl, \; eewl, \; dl, \; nwl, \; rwl, \; gsl;$$

where tn is the terse name, $as_1; \ldots; as_n$ is a list of "argument specifications," s is a skeleton, ewl is a list of "end words," eewl a list of "exclusive end words," dl a list of "delimiters," nwl a list of "noise words," rwl a list of "repeat words," and gsl a list of "generated symbols."

A terse name may be anything that the Editor recognizes as an identifier (see ESD-TR-66-653, Volume I, Section V). Only the name of the terse is required in the terse definition; everything else is optional.

Examples of Simple Terses:

```
DT BATMAN,,  (BRUCE WAYNE, MILLIONAIRE);
DT DISTANCE, (CITYLOC), (GCD(AIRFIELD(LOCATION),
CITY(CITY LOC)));
```

Argument Specifications

After TAP recognizes the name of a terse in an input message, its next job is to scan off the arguments of the terse. Guidelines for scanning off the arguments are prepared for TAP by the definer of the terse. Some of these guidelines are contained in the "argument specifications." Each argument specification has the following form:

$$an, \; xkwl, \; ikwl, \; mar$$

where an is the argument name, xkwl is a list of "exclusive key words," ikwl is a list of "inclusive key words," and mar is a "missing argument replacement." Only the argument name is required.

Argument Name

The argument name may be anything that the Editor recognizes as an identifier. The argument name is used as a dummy atom within the skeleton and replaced by the argument itself when the terse is expanded.

Example

Terse Definition:*

DT MUL, (A;B;C), (L(U),A; *(U), B; ST(U), C;);

Typical Expansion:

MUL 2, X, X + 1;

↓

L (U), 2; *(U), X; ST(U), X + 1;

Exclusive Key Word List

An exclusive key word list is a sequence of identifiers, each of which is to serve as an exclusive key word for the argument being specified. The occurrence of an exclusive key word in the input message being processed by TAP signals that the next atom is the beginning of the argument with which this exclusive key word is associated. The exclusive key word itself is not part of the argument.

"L(U)," "(U)," and "ST(U)" happen to be IBM 7030 instructions meaning load,multiply,and store, but TAP is oblivious to this fact.

Example

Terse Definition:

DT MUL, (A, MULTIPLICAND; B, MULTIPLIER; C, PRODUCT),

   (L(U), A; *(U), B; ST(U), C;);

Typical Expansions:

MUL MULTIPLICAND 2 MULTIPLIER X PRODUCT X + 1;

               ↓

   L(U), 2; *(U), X; ST(U), X + 1;

MUL PRODUCT X + 1 MULTIPLICAND 2 MULTIPLIER X;

               ↓

   L(U), 2; *(U), X; ST(U), X + 1;

MUL MULTIPLIER X, 2, X + 1;

               ↓

   L(U), 2; *(U), X; ST(U), X + 1;

Terse Definition:

DT MUL, (A;B, MULTIPLIER BY; C),

   (L(U), A; *(U), B; ST(U), C;);

Typical Expansions:

MUL P MULTIPLIER Q, ZNAME;

               ↓

   L(U), P; *(U), Q; ST(U), ZNAME;

MUL BY X **2, P, R;

             ↓

   L(U), P; *(U), X**2; ST(U), R;

Inclusive Key Word List

An inclusive key word list, like an exclusive key word list, is a sequence of identifiers, each one of which is to serve as an inclusive key word for the argument being specified. The occurrence of an inclusive key word in the input message being processed by TAP signals that this atom is the beginning of the argument with which this inclusive key word is associated. The inclusive key word itself is part of the argument.

Example

Terse Definition:

DT DOUBLE, (B,,MULTIPLIER BY;C), (MUL TWO, B,C);

Typical Expansions:

DOUBLE X, 2X

↓

MUL TWO, X, 2X

DOUBLE 2X BY X, 2X

↓

MUL TWO, BY X, 2X

Missing Argument Replacement

A missing argument replacement is a parenthetically well-formed string which is used in place of an argument if TAP fails to find the argument in the input message.

Example

Terse Definition:

DT MUL, (A;B;C,,,TEMP), (L(U),A;*(U),B;ST(U),C;);

```
        Typical Expansions:
        MUL  I,J,K ;
              ↓
        L(U),I; *(U), J; ST(U), K;

        MUL  I,J;
            ↓
        L(U), I; *(U), J; ST(U), TEMP;
```

Skeleton

A skeleton is the parenthetically well-formed string that is to
be substituted for the terse name and its arguments in the input message.
The skeleton contains dummy atoms which are replaced by the arguments to
the terse, and may contain "generated symbols" (below).

End Word List

An identifier called an end word is used to signal TAP that the
scope of a terse has ended. When TAP has completed the expansion of the
terse, it will resume processing with the atom that follows the end word.
A list of end words may be specified for a terse. If no end word is
specified, ";" is assumed as an end word (";" has been used in all the
examples up to this point). That portion of the input message which begins
with the name of a terse and ends with its end word is known as a "terse
form."

A terse that has no arguments does not need an end word. If it
is desired to have an end word for a terse with no arguments, then the
terse definition should contain an argument specification, but the argument
name should not appear in the skeleton.

Example

```
Terse Definition:
DT MUL, (A;B;C,,,TEMP), (L(U),A;*(U),B;ST(U),C;),
    STOP / . ;

Typical Expansions:
MUL I, J, K /
            ↓
L(U), I; *(U), J; ST(U), K;

MUL I, J . K
            ↓
L(U), I; *(U), J; ST(U), TEMP; K
```

Exclusive End Word List

An exclusive end word, like an end word, is used to signal TAP that the scope of a terse has ended. It differs from an end word in this way: when TAP has completed the expansion of the terse, it will resume processing with the exclusive end word itself.

Example

```
Terse Definition:*
DT MUL, (A;B;C), (L(U), A; *(U), B; ST(U),C;), (;),
    NOP;

Typical Expansion:
MUL I, J, K NOP; MUL M, N, P NOP;
                ↓
L(U), I; *(U), J; ST(U), K; NOP; L(U), M; *(U), N;
    ST(U), P; NOP;
```

---

*The semi-colon in parenthesis specifies ";" as an end word. If ",," follows the argument specifications, the null string is declared as the end word and the default case of ";" does not take effect. More about this later.

Delimiter List

A delimiter is an identifier that is used to signal the end of
an argument. The delimiter itself is not part of the argument. A list
of delimiters can be specified for a terse. If no delimiters are specified
for a terse "," is assumed ("," is the only delimiter that has been used
in the examples so far). Of course, end words, exclusive end words, and
repeat words (below) also signal the end of an argument.

Example

Terse Definition:

DT MUL, (A;B;C), (L(U), A; *(U), B;ST(U), C;), (;),,
   / ;

Typical Expansion:

MUL I / J / K;
          ↓
L(U), I; *(U), J; ST(U), K;

Noise Word List

A noise word is an identifier which TAP ignores completely during
the expansion of a terse. Noise words can be used by those people who
insist on making believe that computers understand English. A list of
noise words can be associated with a terse.

Example

Terse Definition:

DT MUL, (A;B;C), (L(U), A; *(U), B; ST(U),C;), (;),,(,),
   CRASH RATTLE HISS CRUMP;

Typical Expansion:

```
MUL  CRASH  CRASH  RATTLE  I,  HISS  RATTLE
        RATTLE  CRASH  J  HISS  HISS, CRASH  RATTLE
        RATTLE  CRASH  CRASH  CRUMP  K;
                  ↓
L(U) , I;  *(U) , J;  ST(U) , K;
```

Repeat Word List

A repeat word is an identifier that is used whenever it is desired
to supply more than one set of arguments at one occurrence of the name of
a terse. A repeat word signals TAP that the end of the current set of
arguments has occurred and the next set of arguments begins with the next
atom. A list of repeat words can be associated with a terse.

Example

Terse Definition:

```
DT MUL  (A;B,,,TEMP;C,,,TEMP) , (L(U) , A;*(U) , B; ST(U) ,
     C;) ,  (;) ,,(,) , AND ALSO;
```

Typical Expansions:

```
MUL I, J, K AND M, N, P;
              ↓
L(U) , I;  *(U) , J;  ST(U),  K;  L(U) , M;  *(U) , N;  ST(U) , P;

MUL I, J AND L ALSO P, Q, R;
                    ↓
L(U) , I;*(U) , J;  ST(U),TEMP;  L(U) , L;  *(U) , TEMP;
     ST(U) , TEMP;  L(U) , P;  *(U) , Q;  ST(U) , R;
```

Generated Symbol List

A generated symbol is an identifier which is declared in the
generated symbol list and occurs as one or more atoms in the skeleton.

When the terse is expanded, the generated symbol is replaced at all its occurrences by another identifier which is unique to this particular expansion of the terse.

Example

Terse Definition:

DT DOUBLE.IF.NEGATIVE, (A),

    (IF A GE Ø ; GO X ;

       A = A * A;

    X :),

(;) ,, (,) ,,, X;

Typical Expansion:

DOUBLE.IF.NEGATIVE I;

      ↓

IF I GE Ø; GO GS19;

I = I*I;

GS19:

Stripping and Parenthetically Well-Formed Strings

Sometimes the terse-definer might wish to have a comma, a semicolon or other key word as a part of an argument. A facility to allow this is incorporated into TAP.

A parenthetically well-formed string (or pwfs) is a string of atoms such that in no initial segment the number of right parentheses exceeds the number of left parentheses, and the total number of left parentheses is equal to the total number of right parentheses.

The following strings are parenthetically well-formed:

(((()))),   a(b+c),   (),   ()(),   (()),

The following are not parenthetically well-formed:

),   ((()()),   a+b)(,   (c+(d-e),   )(,

A parenthesized parenthetically well-formed string (or ppwfs) is a string that begins with a left parenthesis and ends with a right parenthesis, and has the property that if the first and last atoms (the just mentioned left and right parentheses) are removed, then the result is a parenthetically well-formed string. The following are ppwfs's:

(()),   ((()())()),   (a+b),   ( ),

The following are not ppwfs's:

()(),   a+b+(c-d),   (a*b c)e,   ((()()())())(),

"Stripping" is the process of taking a string and, if and only if it is a ppwfs, removing the left parenthesis at the beginning and the right parenthesis at the end.

The following table gives some examples of stripping:

string	(())	()()	a	()	a(b)	(,)
stripped string	()	()()	a		a(b)	,

Note that in the fourth example the result of stripping is a null string.

All terse arguments are stripped. This is to allow commas, semicolons and other key words to be arguments of a terse (or parts of arguments).

If the argument of a terse would normally be a ppwfs, it must be surrounded by another pair of parentheses to nullify the effect of stripping. It is standard practice when defining a terse that expands into other secondary terses to parenthesize the arguments of the secondary terse when they are the arguments of the main terse.

In summary, in regard to parentheses there are two things that must be remembered when defining terses:

1. Key words that are nested in parentheses are not recognized by TAP.

2. Arguments that begin with a left parenthesis and end with a right parenthesis lose their outermost parentheses when they are inserted into a skeleton.

Example

      Terse Definition:

      DT MUL (A;B;C), (L(U),A; *(U), B; ST(U), C;);

      Typical Expansion:

      MUL   I, (J;*(U), J), P2;

              ↓

      L(U), I;*(U), J; *(U), J; ST(U), P2;

A More Precise Explanation of Argument Fetching

    When TAP expands a terse, it has the job of supplying a value
to each of the argument names. This is done in the following manner:

    First, all those arguments in the terse form which are identified
by key words are assigned to the argument name associated with the key
word.

    Second, the remaining arguments in the terse form are assigned
to those argument names which have not yet been assigned values. The
first unidentified argument in the terse form is assigned to the first
declared argument name without a value, and so on.

    Third, those argument names which still have no values, are
assigned their declared missing argument replacements as values.

    It should be noted that an argument which is the null string is
not the same as no argument. A null argument is generated when one delimiter
is followed by another with nothing in between, or when an exclusive key
word is followed immediately by a delimiter or an exclusive key word for
another argument. If no argument appears then the missing argument replace-
ment is used, but if a null argument appears the value assigned to the
argument name is the null string.

    Another subtlety in TAP concerns multiple key words for the
same argument. When a key word for an argument is followed by another
key word for that argument and TAP has not yet finished scanning off the

argument, the key word is treated as a noise word if it is an exclusive key word, or as just another atom of the argument if it is an inclusive key word. Two entirely separate occurrences of key words for the same argument will result in undefined operation of TAP.

Creating More Sophisticated Skeletons

Sometimes it is desirable in a terse definition to generate one skeleton segment if a certain condition is true, and to generate another skeleton segment otherwise. This is one application of the conditional sequence actor.

The Conditional Sequence Actor (CS)

The input format expected by the conditional sequence actor is

$$CS \ c, \ (st), \ (sf);$$

where c is either TRUE, or FALSE, or a STEP expression which when evaluated will have a value of TRUE or FALSE; st is the skeleton to be generated if c is TRUE; and sf is the skeleton to be generated if c is FALSE.

The MISSING Actor

The input expected by the missing actor is

$$MISSING \ (s)$$

where s is a string of atoms which may be null. If the string is null, the MISSING actor and its argument are replaced by FALSE. Otherwise they are replaced by TRUE. The MISSING actor may be used to detect the presence of a terse argument.

Terse Definition:

DT   MUL   (A;B, BY;C, INTO),

   (CS MISSING (A), (L(U), A;), (); *(U), B;

    CS MISSING (C), (ST(U), C;), ();),(;),,(,),,AND;

Typical Expansion:

MUL   I   BY   J   AND   BY   K   AND   BY   L   INTO   X   ;

                     ↓

L(U), I; *(U), J; *(U), K; *(U), L; ST(U),X;

The TRANS Actor

TRANS is used to enable actor and terse names to appear in the output string without expanding. The input format for TRANS is:

TRANS   (s)

or

TRANS a

where s is a string and a is an atom. TRANS expands into the string or atom.

Example

TRANS   (DT)

    expands into

DT

The STEP Actor

The step actor is used to execute a STEP expression in the input message at the time the STEP actor is recognized. The STEP

75

actor input format is of the form

STEP e

where e is an expression.

DT Is Really A Terse

Surprise! The thing we've been using all along called DT is really a terse. Like many terses, it has exclusive key words defined for it. In particular,

```
NAME     is an exclusive key word for terse name,
ARGS      "   "       "       "    "   "  argument specifications,
NAME      "   "       "       "    "   "  argument name,
XKEY      "   "       "       "    "   "  exclusive key word list,
IKEY      "   "       "       "    "   "  inclusive key word list,
MAR       "   "       "       "    "   "  missing argument replacement,
SKEL      "   "       "       "    "   "  skeleton
END       "   "       "       "    "   "  end word list,
XEND      "   "       "       "    "   "  exclusive and word list,
DELIMITER "       "       "    "   "  delimiter list,
NOISE     "   "       "       "    "   "  noise word list,
REPEAT    "   "       "       "    "   "  repeat word list,
";" is the end word for DT, and "," is the delimiter for DT.
```

It is now possible to define terses taking advantage of our knowledge of terses.

Example

```
DT SKEL    (CS MISSING (A), (L(U), A;), (); *(U), B;
           CS MISSING (C), (ST(U), C;), ();)

   NAME    MUL
   REPEAT  AND
   ARGS       (NAME A; NAME B XKEY  BY;
              XKEY INTO NAME C);
```

76

ACTOR DEFINITIONS

Actors can be defined by input messages to TAP in the following
format:

DACTOR an (e);

where an is the actor name, and e is a STEP expression. When TAP
encounters an occurrence of the name of the actor in an input message
it evaluates the expression.

In general, there are no rules for writing actors.

A Sample Actor: CS

CS (conditional sequence) is the actor which is used to decide
which of two skeletons to output.

CS has three implicit arguments: a condition and two skeletons.
When CSACTOR, the subroutine that is associated with CS through the STEP
expression (CSACTOR), is called, it appeals to GETAPATOM to get the next
tapatom. This might be TRUE or FALSE or a left parenthesis. All other
cases cause an error message to be output. If it is a left parenthesis
the input string is an s-expression, and QLOTSI and EVAL are called to
evaluate it. If the result is neither TRUE nor FALSE then CSACTOR
indicates an error. In any case, CSACTOR now knows the condition and,
therefore, which skeleton to output.

GETAPATOM is called and its value is checked to see that it is the comma that is required after the condition. If the condition were FALSE then the subroutine STCOMAS (STep to COMma or semi-colon on unmatched right parenthesis in Atom Source) is called to step over the first skeleton. GETATOM is called. The current tapatom should still be a comma. GETAPATOM is then called and the next atom should be a left parenthesis. Everything after this left parenthesis up to the matching right parenthesis is put into a stream. It is read from the atom source so that no expansion will take place yet. This stream is then prefixed to the push down stack ASPS. CSACTOR then reads to the semicolon at the end of the statement by calling the subroutine STCOMAS twice and checking to make sure that the current atom is a semicolon. CSACTOR then returns.

DT Is Really An Actor (Almost)

Surprise again! When the terse DT is expanded, it generates the actor DEFINETERSE. Just for the record, the input format for DEFINETERSE is as follows:

$$\text{DEFINETERSE } t_n(as_1;\ldots;as_n), (s), \text{ewl, eewl, dl, nwl, rwl, gsl};$$

where t_n is the terse name, $as_1;\ldots;as_n$ is a list of argument specifications, s is a skeleton, ewl is a list of end words, eewl a list of exclusive end words, dl a list of delimiters, nwl a list of noise words, rwl a list of repeat words, gsl a list of generated symbols.

An argument specification is of the form

an, xkwl, ikwl, mar

where an is the argument name, xkwl is a list of exclusive key words, ikwl is a list of inclusive key words, and mar is a missing argument replacement. No default cases are supplied for missing arguments to DEFINETERSE.

78

PRE-PROCESSING VS CO-PROCESSING

TAP can operate as either a "pre-processor" or a "co-processor." As a pre-processor, TAP processes an entire input message at once. As a co-processor TAP processes only enough of the input message to generate one atom of output on each appeal. In co-processing, mode control is passed back and forth between TAP and its co-processor. The co-processor thinks of TAP as its input subroutine, and TAP thinks of its co-processor as its output subroutine.

The difference between pre-processing and co-processing concerns actors. Actors are of one of two types: those actors which are concerned with general string manipulation and the internals of TAP, and those which are peculiar to a particular co-processor. The first kind may be used in either pre-processing or co-processing mode, and the second only in co-processing mode.

The definition of all co-processing actors must be preceded by the word COPROCESSOR.

SECTION IV

STEP

INTRODUCTION

STEP language is one of several languages provided by the
C-10 system for building procedures. STEP is a simple language
syntactically. A STEP procedure consists basically of appeals to
C-10 procedures in a prefix functional notation. STEP procedures
may be stored and executed as STEP procedures (using the STEP
interpreter) or translated into AUTOCODER procedures (using the
STEP compiler).

STEP language is based on the LISP system. LISP is
commonly thought of as a list of processing system but, in fact, it
offers much more. Our task in building the STEP language has
been to extract from LISP those things which are useful in a
general system built to manipulate many different types of data
structures, leaving behind those things concerned only with
lists. Readers familiar with LISP will notice the disappearance
of some things. In STEP there is no intrinsic use of CAR, CDR,
and other list functions; no APPLY and no EVALQUOTE.

NAMES AND THE ASSOCIATION LIST

A name is a thing that we use to denote something else. "Pencil" denotes the thing I am writing with; "Elaine" denotes the secretary who will eventually type this sentence; " "pencil" " denotes the first word of this sentence. The reader is undoubtedly familiar with the use of names.

An interesting thing about names is that although the rules for building them out of our alphabet allows for great variety, we tend to use the same names over and over. But despite our over-use of names, we are seldom confused. To illustrate, imagine a girl named Sally who has a husband named Harry, a father named Ted, and a cousin named Elaine. In the "context," or environment of her home, discussing laundry detergents with her mother, a casual reference to Harry will be auto-matically associated with her husband, and a casual reference to Ted will be automatically associated with her father. If Sally goes alone to a PTA meeting that evening and meets a man named Harry and another man named Ted, reference to Harry and Ted will be automatically associated with the new acquaintance throughout the evening. If she has not met someone named Elaine, references to Elaine will still be associated with her cousin. But when Sally returns home and the context of the PTA meeting is left behind, Harry and Ted are once again husband and father. It is almost as though Sally has a list of names and an associated list of denotations in her head. Prior to the PTA meeting, a section of this list looks like this:

```
ELAINE  =  cousin
HARRY   =  husband
TED     =  father
```

At the PTA meeting, the list looks like this:

```
ELAINE  =  cousin
HARRY   =  husband
TED     =  father
```

```
HARRY   =  man who sat on my left
TED     =  man who offered me a ride home
```

When Sally returns home the new denotations associated with
Harry and Ted are erased and the old ones restored:

```
ELAINE  =  cousin
HARRY   =  husband
TED     =  father
```

As a description of the mechanism that people use to keep
names straight, this discussion falls a little short. But as a
description of the mechanism that <u>STEP</u> uses to remember names, it
happens to be fairly accurate.

During the operation of one or more STEP procedures,
the denotations of all names are remembered with an association
list. The association list is sometimes referred to as the "associa-
tion stack," or the A-stack. The denotation of a name is frequently
referred to as the "value" of the name.

When the mechanism which interprets STEP procedures is
initialized, the following names are entered in the A-stack:

Name	Value
NIL	nil
TRUE	true
T	true
FALSE	false
F	false

Throughout the execution of a procedure the STEP interpreter
typically enters and exits several different "contexts." A new
context is encountered each time an appeal is made to a procedure
from within a procedure, each time a "PROG form" is encountered, and
each time an "autonomous procedure name" is encountered (these forms
are explained below). When a new context is entered, a line is
drawn across the A-stack and the names associated with the new
context are entered in the A-stack. The value associated with a
name is always located by searching the A-stack from the new end.
Thus a name which is entered in the A-stack more than once is
always associated with its newest value. A context is exited when-
ever a procedure returns to the procedure which appealed to it,
or whenever the end of a PROG form or autonomous procedure name is
reached. When a context is exited, all the entries in the A-stack
made for that context are erased (all the entries back to the line
that was drawn across the A-stack when the context was entered).

The A-stack is used throughout STEP to associate names with
values. The reader should keep this mechanism in mind as he reads
the following material.

83

EXPRESSIONS

An expression is the basic structure of STEP language. When executed by the STEP interpreter, an expression produces a value. The execution of an expression may also have some "side effects." An expression, by definition, is one of the following things:

> number
> literal
> name
> ATOM form
> QUOTE form
> procedure form
> IF form
> COND form
> PROG form
> GO form
> RETURN form
> ASSIGN form
> DEFINE form

Numbers and Literals

A number is anything that the Editor accepts as either an integer or a floating point number. For all practical purposes an integer is a string of digits not exceeding nine characters in length; and a floating point number is a string of digits with a decimal point either prefixed, suffixed, or imbedded, or one of these possibilities followed by E and an integer exponent (which may be signed with + or -). The value of a number is obvious.

A literal is anything that the Editor accepts as a literal, essentially a character string surrounded by quotes.

Examples

> 'AB/C'
> .32
> 8
> 7.956E-4

Names

A name is anything that the Editor accepts as an identifier.
Roughly speaking, a name is a string of letters and digits not ex-
ceeding 27 characters in length, which may contain embedded periods
if at least one letter precedes the first period. A name may contain
spaces and other punctuation only if it is surrounded with backslashes.

The value of a name is the value associated with the name in
the association list (or A-stack) when the name is encountered by
the STEP interpreter. If the name is entered more than once in the
A-stack the "newest" associated value is used. If the name is not
entered in the A-stack, the value of the name is undefined (a crock
with type code U).

Recall that the names "NIL", "TRUE", "T", "FALSE", "F" are
initially entered in the A-stack.

Examples

```
        X
        Y.2
        NEWYORK
       \NEW YORK\
        LABEL
        NIL
        T
```

ATOM Form

An ATOM form is written

(ATOM a)

or

ATOM a

85

where a is an atom. An atom is one quantum of output from the Editor. That is, an atom is either a number or a name. The ATOM form is one way in STEP language of allowing names to denote themselves. The value of an ATOM form is the atom within it. If the atom is a name, it is not looked up in the A-stack.

Examples

ATOM form	Value
(ATOM A)	A
ATOM \NEW YORK\	NEW YORK
(ATOM 18.4)	18.4
ATOM LABEL.X	LABEL.X

QUOTE Form

A QUOTE form is written

(QUOTE a)

or

QUOTE a

where a is an atom (a number or a name) or a parenthetically well-formed string. The QUOTE form is another way in STEP language of allowing names to denote themselves. The value of a QUOTE form is a stream pointer to the atom or parenthetically well-formed string within it (and not the thing itself).

Examples

QUOTE Form	Value
QUOTE 6	stream pointer to 6
(QUOTE \NEW YORK\)	stream pointer to NEW YORK
QUOTE 18.E-7	stream pointer to 18.E-7
(QUOTE (X(Y)))	stream pointer to (X(Y))

86

Procedure Form

A procedure form is written

$$(pn \quad a1 \ ... \ an)$$

where pn is the name of a C-10 procedure and a1 ... an are the n arguments supplied to the procedure.

A procedure appealed to from a procedure form can be another STEP procedure, an AUTOCODER procedure, or a PROFILE procedure.

A procedure can have from zero to ten arguments. The number of arguments a procedure has is fixed (the same for every appeal). Each argument to a procedure is an expression. Thus each argument to a procedure may be itself an appeal to a procedure.

The value of a procedure form is the value returned by the procedure when the procedure is executed.

A procedure name within a procedure form is the only kind of name in STEP language which is not associated with a value in the A-stack. The STEP interpreter recognizes a procedure name by the context in which it occurs, and locates the associated procedure through C-10 mechanisms external to the STEP interpreter.

Examples

```
(ADD 2  2)
(ADD (ADD 2  2) 2)
(PROG2 (QUOTE XYZ) 26.4E-7)
(ADD 7 (SUB 18 3))
```

<u>Form Forms</u>

In the expression

(ADD 7 (SUB 18 3))

the second argument to the procedure ADD is an appeal to the procedure SUB. This particular expression is evaluated by the STEP interpreter by appealing to the procedure SUB first, which returns a value of 15; then appealing to the procedure ADD with the arguments 7 and 15. Conceivably, this expression could be evaluated by appealing to ADD first, and appealing to SUB second (when ADD discovers that its second argument is not simple). In the case of this example, the evaluation might proceed either way. But there are some cases where it is desirable <u>not</u> to have the arguments of the procedure evaluated before the appeal to the procedure. For example, it is desirable to call some frequently used procedures with names as arguments without the necessity of prefixing each name with "ATOM" or "QUOTE" (unless this is done the associated <u>value</u> of the name is supplied to the procedure). It is possible to define procedures in STEP language which will not have their arguments automatically evaluated before they are called. Such a procedure is called a <u>form</u>. A procedure which is not a form is called a <u>function</u>.

A STEP language procedure which is to operate as a form must be declared as a form when it is defined (this is explained below). A form always has either no arguments or exactly one argument. When a form is appealed to from another STEP procedure, this argument is a stream pointer to the STEP internal representation of whatever follows the name of the form and precedes the right parenthesis matching the left parenthesis preceding the name of the form. That is, if XFORM is

the name of a form, when XFORM is appealed to from the expression

<div style="text-align:center">(XFORM (ADD 6(SUB 3 1)) XYZ NIL)</div>

the one argument supplied to XFORM is a stream pointer to the internal representation of

<div style="text-align:center">(ADD 6 (SUB 3 1)) XYZ NIL</div>

Wnat,the reader may ask, does one do with such a stream pointer?

The reader may follow the following rules which are guidelines to the accessing and evaluation of form arguments.

The single argument to a form is segmented into pieces by the STEP interpreter. Each piece is either a name, a number, or something surrounded by parentheses. The single argument

<div style="text-align:center">(ADD 6 (SUB 3 1)) XYZ NIL</div>

is segmented into the individual pieces

<div style="text-align:center">(ADD 6 (SUB 3 1))
XYZ
NIL</div>

The argument to the form is a stream pointer to the first segment. An appeal to the procedure NXTGRP with the pointer to the first segment will produce a pointer to the second segment. Another appeal to the procedure NXTGRP with the pointer to the second segment will produce a pointer to the third argument, and so on.

89

If A is the argument to a form, A is a pointer to the first argument segment,

<div align="center">(NXTGRP A)</div>

is a pointer to the second segment, and

<div align="center">(NXTGRP (NXTGRP A))</div>

is a pointer to the third argument segment. NXTGRP returns a value of nil when there is no next segment.

If any or all of the segments of the argument to a form are STEP expressions, the value of the expressions may be obtained by appealing to the procedure EVAL with a pointer to an argument segment as argument.

It should be noted that declaring a procedure a form is one way of getting around the restriction that a procedure must have a fixed number of arguments. If the argument segments of the single argument to a form are thought of as individual arguments, then in this sense a form can have a variable number of arguments. If PLUS is a form, one may write

<div align="center">(PLUS 3 (SUB 7 6) 8 16 42 14)</div>

and expect PLUS to add all of its "arguments" together at once. Actually, PLUS is supplied with only one argument. PLUS obtains the values of its "arguments" by appealing to NXTGRP and EVAL. If A is the argument to PLUS in the above example,

<div align="center">

(EVAL A) = 3
(EVAL (NXTGRP A)) = 1
(EVAL (NXTGRP (NXTGRP A))) = 8, etc.

</div>

Autonomous Procedure Names

Autonomous procedure names are names of procedures which contain within them the definition of the procedure they name. Autonomous procedure names are rare in programming languages, but have been used by mathematicians and engineers for a long time. When an engineer says "the function sine" he uses the name "sine" to denote the common trigonometric function, the definition of which everyone knows. But when an engineer says "the function $X^2 + 1$" he uses the name "$X^2 + 1$" not only to denote the function, but also to define it. "$X^2 + 1$" denotes the function with value equal to the square of the argument to the function plus one. One difficulty in using autonomous procedure names is that it is not always clear what the arguments to the procedure are supposed to be. If "$aX + b$" is an autonomous procedure name, are "a", "X", and "b" all names of arguments? Is only "X" the name of an argument? In normal conversational usage, it is usually obvious which of the symbols within an autonomous procedure name represent arguments, but when autonomous procedure names are used within STEP language, the list of argument names must be spelled out explicitly.

An autonomous procedure name in STEP language is written:

$$(\text{FUNCTION } (n_1 \ldots n_k)\ e)$$

or

$$(\text{FORM } (n)\ e)$$

depending on whether the procedure is to be a function or a form. $n_1 \ldots n_k$ is a list of k names which are used as names of arguments to a function (if any); n is a name which is used as the name of the argument to a form (if it has one); and e is an expression which defines the value of the procedure.

An autonomous procedure name can be used wherever a
regular procedure name can be used. In particular it can be used in a
procedure form. Note that the autonomous procedure name itself does
not represent a value and is not one of the things previously listed
as a possible expression.

<u>Examples of Autonomous Procedure Names</u>

(FUNCTION (X) (PLUS (TIMES X X) 1))

(FUNCTION (A B) (TIMES (EXP A B) A))

(FUNCTION ( ) TRUE)

<u>Examples of Autonomous Procedure Names Used in
Procedure Forms</u>

((FUNCTION (X) (PLUS(TIMES X X) 1)) (SQRT 47))

((FUNCTION (A B) (TIMES (EXP A B) A)) (SIN (TAN 45)))

((FUNCTION ( ) TRUE))

((FUNCTION (X) (PLUS (TIMES X X) 1)) ((FUNCTION (X) (PLUS
           (TIMES X X) 1)) (SQRT 47)))

It is easy to imagine how a procedure form which con-
tains an autonomous procedure name as the procedure name is evaluated by
the STEP interpreter. The evaluation of such a procedure form constitutes
a new "context" as discussed in the Names and the Association List. First
the arguments to the procedure are evaluated. Then a line is drawn across
the A-stack and the names following "FUNCTION" or "FORM" in parentheses are
entered in the A-stack and associated with the values of the arguments.
Then the expression within the autonomous procedure name is evaluated.
The value of the expression within the autonomous procedure name becomes
the value of the procedure form. Finally the A-stack is restored to its
former status.

<u>Another Example</u>

((FUNCTION (X) ((FUNCTION (X) (TIMES X X)) (TIMES X X))) 2)
when evaluated yields a value of 16.

IF Form

An IF form is written

$$(IF \; e_1 \; e_2)$$

where e_1 and e_2 are expressions. An IF form is evaluated by evaluating
the first expression, and then if and only if the value of the first
expression is true, evaluating the second expression. The value of the
IF form is the value of the second expression if the value of the first
is true; the value of the IF form is nil if the value of the first ex-
pression is not true. This may sound a little silly. This is because
the IF form is usually used not to generate a value, but to execute
(evaluate) the second expression conditionally, and thereby cause the
"side effects" of evaluating the second expression to occur.

Examples

(IF (EQUAL A 6) (PROG5))

If the value associated with the name "A" in the A-stack is
6, the procedure PROG5 is executed. The actual value of the IF form
is whatever value the procedure PROG5 returns if A equals 6; the value
of the IF form is nil otherwise.

(IF FALSE TRUE)

The value of this IF form is nil.

(IF (IF TRUE TRUE) FALSE)

The value of this IF form is false.

(IF 6 8)

The value of this IF form is nil.

93

COND Form

A COND form is written

$$(\text{COND } (e_{11} \ e_{12})(e_{11} \ e_{22})\ldots(e_{n1} \ e_{n2}))$$

where $e_{i1} \ e_{i2}$, $1 \leq i \leq n$, is a pair of expressions. The COND form is evaluated as follows: The first expression in the first pair of expressions is evaluated. If its value is true, the second expression in the first pair of expressions is evaluated and its value becomes the value of the COND form. If the value of the first expression is not true, the process is repeated with the second pair of expressions, and so on. If none of the values of the first expressions are true, the value of the COND form is nil.

Examples

(COND ((EQUAL A 1) 3) ((EQUAL A 2) 6)
      ((EQUAL A 3) 9))

This COND form has a value of three times A if A is 1,2, or 3; nil otherwise.

(COND (FALSE TRUE))

This COND form has a value of nil.

PROG Form

Thus far, STEP language does not look much like a programming language. The only things described so far have been a mechanism for associating values with names, a mechanism for appealing to procedures, and a mechanism for conditionally evaluating expressions. But, in fact, together with a mechanism for defining new procedures these things would comprise a completely adequate programming language. For example, the reasonably complicated procedure QUOTIENT could be defined by this autonomous procedure name:

```
(FUNCTION (A B) (COND ((GREATER A B) Ø)
                      ((EQUAL A B)   1)
                      ((GREATER B A)
                       (ADD 1 (QUOTIENT (SUB B A) A)))))
```

However, despite the adequacy of the STEP language described thus far, the PROG form is defined to make STEP language more like conventional programming languages, thus making STEP more palatable to conventional programmers and possibly effecting some efficiencies. The PROG form introduces STEP analogs of several common programming concepts which have not yet been mentioned: the concept of a "label," the concept of a "branch instruction," and the concept of "temporary storage."

A PROG form is written

$$(PROG \ (n_1 \ldots n_j) \ e_1 \ldots e_k)$$

where $n_1 \ldots n_j$ is a list of names and $e_1 \ldots e_k$ is a list of expressions.

The names $n_1 \ldots n_j$ serve as "temporary storage" in the following way. The scope of a PROG form is a new "context" to the STEP interpreter, in the manner discussed in the Names and the Association List. When the PROG form is interpreted a line is drawn across the A-stack and each of the names in the list of names following "PROG" is entered in the A-stack associated with the value nil. During the execution (evaluation) of the remainder of the PROG form, the values of these names may be referenced and changed, just as temporary storage is changed in a conventional program. When the evaluation of the PROG form is complete, the A-stack is restored to its former status (the names are no longer defined, or resume their previous identity). These names are often referred to as "PROG variables."

Some of the expressions in the expression list $e_1 \ldots e_k$ serve as "labels." The next thing that the STEP interpreter does after entering the PROG variables in the A-stack is to examine each of the expressions in the list $e_1 \ldots e_k$. <u>Each expression which does not begin with a left parenthesis and end with a right one is entered in the A-stack and associated with a value which is a pointer to its location in the PROG form</u>. Of course, there are only two kinds of expressions which do not begin with a left parenthesis and end with a right one: a name and a number. It is not recommended that numbers be used as labels; i.e., they should not stand alone in the expression list $e_1 \ldots e_k$. STEP will not complain of this although some undefined results may occur. The A-stack is restored to its previous status following the evaluation of the PROG form.

After the PROG variables and labels of a PROG form have been entered in the A-stack, the execution (evaluation) of the PROG form continues. The STEP interpreter then evaluates the expressions $e_1 \ldots e_k$, one at a time, proceeding from left to right, throwing away the values of the expressions as they are obtained. The value of a

PROG form is nil, unless a RETURN form is executed (below). The
sequential interpretation of the expressions $e_1 \ldots e_k$ may be interrupted
by the execution of a GO form (below).

Examples of PROG forms will be given following the explanations
of the GO form, the RETURN form and the ASSIGN form.

GO Form

A GO form is written

$$(GO\ n)$$

where n is a name. When a GO form is evaluated, the associated value
for the name is found in the A-stack. This associated value should be
a pointer to a place within a PROG form. If indeed it is, interpretation
is interrupted, and begun again with the label pointed to. If the value
associated with the name is not a pointer, an error message is printed
and the value of the GO form is undefined.

RETURN Form

A RETURN form is written

(RETURN e)

where e is an expression. When a RETURN form is executed within a
PROG form, interpretation of the PROG form ceases, and the value of
the PROG form is the value of the expression within the RETURN form.
When a RETURN form occurs outside a PROG form, interpretation of the
STEP expression being interpreted ceases, and its value is the value
of the expression within the RETURN form.

ASSIGN Form

An ASSIGN form is written

(ASSIGN n e)

where n is a name and e is an expression. An ASSIGN form is used to
change the associated value of a name in the A-stack. Only the most
current entry of the name in the A-stack is changed. The associated
value of the name in the A-stack becomes the value of the expression e.
The value of the ASSIGN form is also the value of the expression e.

Examples

(PROG () (RETURN TRUE))

The value of this PROG form is true.

```
((FUNCTION (A B)
     (PROG (QUOTIENT)
       (ASSIGN QUOTIENT Ø)
       LABEL (IF (GREATER B A) (RETURN QUOTIENT))
       (ASSIGN A (SUB A B))
       (ASSIGN QUOTIENT (ADD QUOTIENT 1))
       (GO LABEL)))
  847
  63)
```

The value of this expression is 847 divided by 63.

```
(EQUAL NIL (PROG(X)
              (ASSIGN X D)
              (ASSIGN D C)
              (ASSIGN C X)))
```

The value of the expression is true. In the course of its
evaluation the values of C and D are interchanged in the A-stack.

```
(PROG () TRUE (IF TRUE (RETURN FALSE)))
```

The value of this expression is nil.

```
(ASSIGN A(ASSIGN B(ADD 6 6)))
```

This expression has a value of 12. It also assigns the
value 12 to the names A and B.

```
(PROG () (IF (EQUAL 3 4) (RETURN TRUE)))
```

This expression has a value of nil.

DEFINE Form

We have seen in earlier discussion how an autonomous procedure
name serves as a name of a procedure and also specifies the algorithm
for determining the value of the procedure. As the name of a pro-
cedure that is to be used again and again, an autonomous procedure
name is a little awkward. The DEFINE form is used to specify a
unique name that is to be used in place of a particular autonomous
procedure name. Upon the execution of a DEFINE form, a new name
is added to STEP's list of procedures, and is associated with the
designated autonomous procedure name. Henceforth, that name can be
used to invoke the procedure thus defined.

99

A DEFINE form is written

$$(\text{DEFINE} \ (n_1 \ apn_1) \ (n_2 \ apn_2) \ \dots \ (n_k \ apn_k) \ )$$

where $n_1 \ apn_1$, $1 \le i \le k$, is a name and the autonomous procedure name that is to be associated with it.

Examples

```
(DEFINE (DOUBLE (FUNCTION (X) (ADD X X)))
        (TRIPLE (FUNCTION (X) (ADD(DOUBLE X) X))))

(DEFINE ( QUOTIENT (FUNCTION (A B)
        (PROG (QUOTIENT)
          (ASSIGN QUOTIENT Ø)
        LABEL (IF(GREATER B A) (RETURN QUOTIENT))
          (ASSIGN A (SUB A B))
          (ASSIGN QUOTIENT (ADD QUOTIENT 1))
          (GO LABEL))))

(DEFINE (PLUS (FORM (A)
        (COND  (( EQUAL A NIL) Ø) (TRUE
          (ADD (EVAL A) (PLUS (NXTGRP A )))))))))
```

A FEW PROCEDURES

The answer to the question "What procedures can be appealed to from a new STEP procedure?" is "any procedure in the C-10 system: any STEP procedure, any AUTOCODER procedure, any PROFILE procedure." There are over 400 such procedures, far too many to describe here. However, for the benefit of the reader who wishes to become familiar with STEP by writing some sample procedures for exercise, a few procedures are listed below to get him off the ground.

100

Name	No. of Args.	Action/Value
ADD	2	Returns sum of arguments, which must have numeric values.
ADD1	1	Returns sum of argument, which must have numeric value, and one.
AND	(form)	Returns logical AND of arguments (true or false) which must have values of true or false.
DIV	2	Returns quotient of arguments, which must have numeric values.
EQUAL	2	Returns equal comparison of arguments (true or false), which must have numeric or literal values.
GREATER	2	Returns greater comparison of arguments (true or false), which must have numeric values.
MUL	2	Returns product of arguments, which must be numbers.
NOT	1	Returns negative of argument, which must be true or false.
NULL	1	Returns true if argument is nil; returns false otherwise.
OR	(form)	Returns logical OR of arguments (true or false) which must have values of true or false.
PRINTC	1	Prints argument on console typewriter; returns nil.
TRACE	(form)	Turns on tracing for all procedures named in argument list.
UNTRACE	(form)	Turns off tracing for all procedures named in argument list.

COMPILING STEP PROCEDURES

     STEP procedures which have been defined using the DEFINE form
can be compiled into AUTOCODER procedures using the STEP compiler.
Unfortunately, it is possible to write procedures in STEP language which
behave differently when they are compiled into AUTOCODER. To make sure
that a STEP language procedure can be compiled successfully into AUTOCODER
these rules of thumb should be followed:

> (1) Don't change the denotations of labels.
> Programs like the following will not work
> when compiled.
>
>   (FUNCTION (A) (PROG (X) (ASSIGN X A) (GO X)))
>
> (2) Don't tamper with the "constants" TRUE, T, FALSE,
> F, NIL.
>
> (3) Don't appeal to procedures in STEP or PROFILE
> unless you intend to compile them also.

THE USES OF STEP LANGUAGE IN C-10

     STEP language is used in the following ways:

> (1) STEP procedures are used as basic building
> blocks of C-10.
>
> (2) STEP expressions can be entered directly
> from the console and evaluated on-line.
>
> (3) STEP expressions may be embedded in PROFILE
> procedures and queries.
>
> (4) STEP expressions are used to define actors to
> TAP, the terse/actor processor.

102

TRACING

A special tracing facility is provided for tracing STEP
procedures which are being interpreted.

Briefly, there are two procedures which are used to turn
tracing on and off: TRACE and UNTRACE. Each of these procedures is
a form and accepts a variable number of arguments, which except for
the first one are names of procedures. The first argument specifies
which trace program is to be used. The STEP trace is probably the most
useful for debugging STEP programs.

Example

(TRACE STEP EXP MUL ADD)

A sample trace, showing the evaluation of the expression

$$X^2 + 2X + 1$$

appears below for X = 4:

```
TRACING (EXP X 2)
ENTER EXP
    WITH 2 ARGUMENTS..
    4
    2
LEAVE EXP
    WITH VALUE..
    16
TRACING (MUL 2 X)
ENTER MUL
    WITH 2 ARGUMENTS..
    2
    4
LEAVE MUL
    WITH VALUE..
    8
TRACING (ADD(EXP X 2)(MUL 2 X))
ENTER ADD
    WITH 2 ARGUMENTS..
    16
    8
LEAVE ADD
    WITH VALUE..
    24
TRACING (ADD(ADD(EXP X 2)(MUL 2 X)) 1)
ENTER ADD
    WITH 2 ARGUMENTS..
    24
    1
LEAVE ADD
    WITH VALUE..
    25
```

SECTION V

MODULAR MACHINE LANGUAGE PROCEDURES

RELOCATABLE SUBROUTINES

Relocatable subroutines in C-10 are both recursive and
dynamically relocatable. To accomplish this, it was necessary to place
some restrictions on their form and use, and to establish a common set
of rules for writing them.

The need for a standard interface for linkage between routines
in a variable word length machine has resulted in the invention of fixed
length fields called crocks, an arbitrary ten characters long, with a
word-mark over the leftmost character.

Three regions containing crocks are defined specifically to
implement the subroutine linkage:

QTEMP*   This symbol defines the <u>left</u> end of a set of
                30 crocks, the temporary storage list. They
                can be addressed by the names QT1 through QT30
                each of which defines the right end of its
                respective crock. Thus, QT1 is QTEMP+9, QT2
                is QTEMP+19, etc. The names QT1 through QT30
                should not be used within the body of subroutines
                but only within EQU statements at the beginning.
                (This makes subroutines easier to change, in
                addition to making the symbols more mnemonic).

---

*Since the assembly system, Autocoder, has no heading or tailing
facilities, system symbols and routine names are conventionally
defined with a prefix Q. To avoid duplication with Q-symbols, the
individual user simply uses symbols which begin with a letter other
than Q.

QARGL   This symbol defines the <u>left</u> end of a set of 10
                 crocks, the argument list. The names QARG1
                 through QARG10 define the individual crocks as
                 described above for QTEMP.

QVALUE   This symbol defines the <u>right</u> hand end of a single
                 crock used for subroutine values.

CALLS AND RETURNS

When one subroutine wishes to call another, it places the
arguments (appropriate in number and form) for the program to be
called (callee) in the argument list, QARGL, and executes a call to the
callee.

Subroutine Control (RESC) first guarantees that the callee is
in memory, making space for it and reading it in if necessary. Callee
has declared, in the first 48 characters of his code, the number of
arguments he expects and the number of temporary registers he will
require. RESC adds these two figures together to determine the
amount of QTEMP that will have to be saved, and saves this number in
its own, private pushdown stack. The arguments are transferred from
QARGL to the corresponding slots* in QTEMP, and each of the additional
requested QTEMPS, immediately following the arguments, is set to a
type code of N.

Thus, a subroutine, on receiving control, will find its n
arguments in the first n slots of QTEMP, (not QARGL). QARGL is
immediately available for use in setting up the arguments for another
call, without destroying the arguments for this one.

---

*field containing a crock.

106

Since the temporary slots requested are assigned to the slots following the arguments, and there are a total of 30 QTEMP slots, it follows that the number of temporary slots in which crocks may be kept is limited to 30 minus the number of arguments.

When the routine has completed its task, it sets up a single crock, QVALUE, with its result, which will be returned to the caller. A limit of a single value for a subroutine has on occasion proved distasteful. Additional crocks have been proposed to augment QVALUE. At this writing, three such crocks have been defined. QSTRVAL is used privately by block management, at times, to pass back additional information which it feels may be useful to its users. QVALUE2 and QVALUE3 have been used by other procedures for a similar purpose.

DYNAMIC SUBROUTINE RELOCATION

Subroutine Loading

We have mentioned earlier that RESC, if it needs to, will make space for, and load a program if it is not already in memory.

Unlike many systems, in C-10 there is no attempt to force the user to forecast his usage of other subroutines by declaring and loading them in advance, as is the case with FORTRAN. Instead, as each subroutine is called for the first time, it is read from disk into the next available space in memory. When the available space is not sufficient to accommodate a subroutine to be loaded, subroutines that have not been used for a long time (i.e., are "oldest") are discarded until enough space is available, the remaining subroutines are consolidated in the program area, and the new subroutine is loaded at the end.

Self Modification

The ability to dynamically move a subroutine after it has
been loaded into core is made possible partly by the relocation
method used and partly by some rules the user must follow.

Subroutines are assembled as though they were to operate at
location $\emptyset$. When RESC places a subroutine into a specific core loca-
tion, it also places that location into X1 (index 1). It is the
responsibility of the user to modify every internal address symbol
in his program by X1. For example:

```
6               16    21
TEMP            EQU   QT1
                B     LOC+X1
LOC             MLC   DATA+X1,TEMP
                CALL  QROUTINE,,'742'+X1
```

where Q-symbols are absolute and need no modification. Note that
every time a branch to LOC is desired, it must be written LOC+X1.
This can be avoided by defining the symbols themselves to include
the X1 reference:

```
TEMP            EQU   QT1
                B     LOC
LOC             EQU   *+X1
                MLC   NAME,TEMP
                B     LOC
                DCW   'NAME4'
NAME            EQU   *-1+X1
```

Note the difference in defining symbols for data and for instructions:

instructions - definition is *+X1 <u>preceding</u> the instructions

data   - definition is *-1+X1 <u>following</u> the data

Index register usage, other than that of X1, is discussed in a later section.

When a subroutine gives up control to call another, it may be discarded and read back in, or it may merely be moved before it again regains control. Temporary storage within the subroutine may or may not be changed on return from the call. Thus, all information which the subroutine wishes preserved across the call must be kept in QTEMP registers reserved for that purpose. Since the QTEMP contents are placed in the pushdown list at the time of a call, and restored on the return, they appear unchanged to the user. Briefly, the two rules discussed above may be summarized as follows:

1. All internal symbols within the subroutine must be modified by X1.

2. Subroutines may not be self-modifying across a call, but should use QTEMP instead.

Recursiveness

The rules and mechanisms required to allow dynamic subroutine relocation provide the capability for any subroutine to call itself recursively. It need make no special provisions to permit the re-entry.

MACROS

Five macros have been provided to be used where applicable in every subroutine.

BEGIN

Every subroutine must begin with the BEGIN macro. The
expansion of the macro supplies the current definitions of all system
symbols, plus the header material which precedes each subroutine.

                         16   21
                         BEGIN rname,args,temps,type

rname   -   is the name of the subroutine being compiled. It
             may be a two to ten character alphanumeric name
             which begins with the character Q.

args   -   is a three digit number* which specifies the number
             of arguments (∅∅∅ to ∅1∅) expected by this routine.

temps   -   is a three digit number* which specifies the number
             of words of temporary storage (QTEMP) required by this
             routine in addition to those used for arguments.

type   -   normally omitted, this field allows the user to
             specify two special kinds of subroutines, forms (F)
             and absolute (A).

             A form is special in that it has only one explicit
             argument. This argument is a pointer to the first
             member of a list of arguments, whose number is indefinite.
             the stream function NXTGRP is used to get the next
             argument of a form. When NXTGRP returns NIL, there
             are no more arguments.

             Absolute routines are rare in C-10. Examples
             include IOCS and RESC itself.

_____

*Leading zeros required.

110

CALL

All subroutine calls must be made with the CALL macro.

```
            16      21
            CALL    rname,value,arg1,arg2,...,argn
```

rname - is the name of the subroutine being called. It is the only required parameter of the CALL macro.

value - is a location into which QVALUE is to be saved upon return to the caller.

arg1,
arg2,...
argn - represent the arguments required by the routine being called. Those specified will be moved by the instruction.

```
            MLC argn,QARGn
```

The user may, of course, set up his own arguments, i.e.,

```
            MLCS QI,QARG1
```

```
            ZA    Q1,QARG1-1
```

Arguments so set up should be omitted from the CALL statement.

RETRN

The RETRN macro may be used whenever control is to be returned to the calling subroutine.

```
            16      21
            RETRN   value
```

111

If value is specified, the instruction

MLC value, QVALUE

is compiled before the return.

RETRN may appear more than once in any subroutine, but should appear at least once.

FIN

The FIN macro has no parameters. It must appear once and only once in every subroutine, positioned as the next to last card of the deck, just before the END card.

```
16
FIN
```

TEARS

The TEARS macro is used whenever an error message is to be output by a subroutine.

```
16        21
TEARS    enum,parl,...,par7
```

The TEARS macro has from one to eight parameters. The first parameter is an error message number and is written n! (an integer followed by an exclamation point). The remaining parameters are either other error message numbers (which are also written n!) or printable crocks. The parameters which follow an error message number are inserted into the error message, if the error messages require parameters; otherwise, they are added to the end of the error message.

Each message specifies whether the "current line of input" is to be printed along with the message or not, or if the decision to print the "current line of input" is to be based on the setting of the global switch QTOGGLE. QTOGGLE is set to 1 if the "current line of input" is to be printed, 0 otherwise.

New messages may be added and old messages may be changed with the help of the subroutine CHANGTEAR.

INDEX REGISTERS

Description

X1: Containing the origin of the currently operating program, it may not be altered by anyone other than RESC.

X2: Reserved for RESC use, it always points to the most recently used position of the pushdown list (QPUSHL) internal to RESC. Only RESC may alter it.

X3-X13: These index registers may be used by the individual subroutines; each one used must be saved before use and restored to its original value before the program returns.

X14,X15 These index registers may be used freely by the individual subroutines. They need not be saved or restored; however, their values are not guaranteed across a call. Thus, in the example,

```
TEMP        EQU        QT5
            ZA         '13'+X1,X10
            CALL       ROUTINE
            MLC        NAME+X10,TEMP
```

the MLC uses the value of X10 established
before the call. If this same sequence had
used X14 in place of X10, X14 might have
changed during the call.

PR-155 NOFLG Option

The PR-155 Autocoder assembly program considers X14 and X15
to be reserved for monitor use, and will compile M (multiply defined
symbol) flags whenever X14 or X15 appear in a program. Since C-10
does not use the PR-155 monitor, and since the compilation is correct
despite the flags, they may be ignored. A method exists, however,
to suppress these flags by expanding the EXEQ card to add the
parameter NOFLG:

```
6          16         21
MON$$      EXEQ       AUTOCODER,,,NOFLG
```

This will suppress only the M-flags caused by the use of X14 or X15;
M-flags for truly multiply defined symbols are not suppressed.

Symbolic Use

It is recommended that the use of index registers 3 to 15
be symbolic, rather than absolute. Autocoder accepts statements
such as

XA EQU X15

as long as the EQU appears <u>before</u> any use of that symbol as an index
register.

114

In particular, it has become a fairly common convention to define X15 as XA, and X14 as XB.

CLOSED SUBROUTINES

In a dynamically relocatable environment, the use of closed subroutines within a subroutine can be somewhat tricky.

The major danger arises when the entire subroutine is moved between a call to and the return from the closed subroutine. Further, the conventional store B-register into the exit instruction does not work at all in the C-10 environment, since the store B-register instruction <u>must not</u> be indexed*, and, on the other hand, references to C-10 instructions <u>must</u> be indexed by X1.

Two techniques are available, one using an index register, the other a QTEMP. The index register method tends to be more useful if parameters or error returns follow the branch to the closed subroutine.

For both methods, it is necessary that the sign of the location into which the B-register is stored be guaranteed positive, since the store B-register stores only into the numeric positions of the addressed field, leaving the zones untouched.

<u>QTEMP Method</u>

```
BREG          EQU QT1-5      Note:  BREG requires a high-
                                    order word-mark.

CLSUB         EQU *+X1

              SBR BREG              Set up BREG

              S   X1,BREG          Compute relative return
              ⟨
```

*AUTOCODER will compile, without flags, such an instruction:
                SBR   X +5+X1
on execution, however, the hardware will ignore the indexing bits.

```
                      {
                      CALL
                      {
                      {
          MLN     BREG,EXIT+5              Store relative return
EXIT      EQU     *+X1
          B       EXIT                    The address compiled
                                          will use X1.
```

If no CALL occurs within the closed subroutine, it can be written:

```
BREG      EQU     QT1-5                   Again, BREG requires a
          {                               high-order word-mark.
CLSUB     EQU     *+X1
          SBR     BREG
          {
          MLC     BREG,EXIT+5
EXIT      EQU     *+X1
          B       EXIT
```

Index Register Method

Assume that the closed subroutine has an error return following the branch, and a normal return seven characters further down:

The call:

```
          B       CLSUB
          B       ERROR                   Error return
          Normal Return
```

The subroutine:

```
CLSUB     EQU     *+X1
          SBR     INDEX
          S       X1,INDEX                Make index relative
```

116

```
        ?
        CALL
        ?
        A          X1,INDEX              Make index absolute

        B          0+X1                  Error return
        ?
        B          7+X1                  Normal return
```

As a general rule, INDEX must be relative whenever a CALL occurs, and may be absolute otherwise.

This method does not use any less space than using QTEMP, since INDEX was probably stored in a QTEMP to make it available here. The free indexes, X14 and X15, cannot be used here since they are not guaranteed across the CALL.

However, if there is no call, X14 or X15 may be used if they are otherwise available:

```
CLSUB      EQU        *+X1

           SBR        X14
           ?
           B          0+X14                 Error return
           ?
           B          7+X14                 Normal return
```

DECK MAKEUP

	6	16	21
	MON$$	JOB	Programmer, project, dept.,room
	MON$$	EXEQ	AUTOCODER,,,NOFLG
		TITLE	Sub-routine name
		HEADR	for the top of each page
		PST	if cross-reference listings desired.
Complete Program		BEGIN	See text for format.
		(Autocoder Statements)	
		FIN	
		END	

117

Multiple programs may be compiled as part of a single job. If the NOFLG option is desired on assemblies other than the first of a job, a card similar to the EXEQ card, but without the "MON$$" or "EXEQ" should be placed before the TITLE card of each job:

            21
            AUTOCODER,,,NOFLG

The conventions for assembling AUTOCODER procedures are outlined in Section II of this volume.

```
          HEADR DWRITE(OWNID)
          BEGINQDWRITE,001,006


*                                  25 APRIL 1966 - LJL
*                                   9 JUNE 1966 - LJL
*
OWNID      EQU  QT1
*
ENDNCPLOC  EQU  QT2
GPTRLOC    EQU  QT3
SCENTRY    EQU  QT4
LCOUNT     EQU  SCENTRY-3
COUNTER    EQU  SCENTRY-7
ENDZPTR    EQU  QT5
GRPPTR     EQU  QT6
FWDLINK    EQU  QT7
           CALL QPOSITION,,OWNID    POSITION TO NEXT GROUP OCCURRENCE
           MLC  QVALUE,FWDLINK
           MLC  QVALUE2,GPTRLOC
           MLC  QVALUE3,SCENTRY
           MLC  QSTRVAL,GRPPTR
           CALL QSEARCHPS,ENDZPTR,GPTRLOC,ENDVAR  GET ENDOFFILE PTR
           MLC  QSTRVAL,ENDNCPLOC  SAVE POINTER TO ENDZPTR IN SC STK
           CALL QPINDEX,,ENTER,GRPPTR,ZERO,,ENDZPTR
*     UPDATE GROUP PTR OF OWNID IN SC STACK
           CALL QPINDEX,,ENTER,GPTRLOC,ZERO,OWNID,ENDZPTR
           BCE  SETSCENT,SCENTRY,N
PUTSCENTRYEQU  *+X1
*     MAKE 9 CROCK ENTRY IN FILE, FIRST ENTRY FOR NEW REPETITION
*     OF THIS GROUP
           CALL QPINDEX,,ENTER,ENDZPTR,ZERO,,SCENTRY
           ZA   LCOUNT,COUNTER
           CALL QPINDEX,,ENTER,ENDZPTR,ONE,,FWDLINK
           MLC  QSTRVAL,ENDZPTR
           BCE  CONTINUE,FWDLINK,N
           CALL QPINDEX,,ENTER,FWDLINK,TWO,,QSTRVAL
CONTINE    EQU  *+X1
           CALL QPINDEX,,ENTER,ENDZPTR,ONE,,GRPPTR
*     ENTER LCOUNT NILS
LOOP       EQU  *+X1
           CALL QPINDEX,,ENTER,QSTRVAL,ONE,,QN
           S    Q1,COUNTER
           BZ   SETECROCK
           B    LOOP
```

```
SETECROCK EQU  *+X1
*      ENTER EOS(END OF STACK) CROCK, LAST ENTRY
           CALL QPINDEX,,ENTER,QSTRVAL,ONE,,EOSCROCK
*      UPDATE ENDZPTR(END OF FILE PTR) IN SC(SUPERCLA) STACK
           CALL QPINDEX,,ENTER,ENDNCPLOC,ZERO,ENDVAR,QSTRVAL
           RETRN
SETSCENT  EQU  *+X1
*      GET 9 CROCK (LEVEL,LCOUNT,PARID) FROM SC STACK
           CALL QPINDEX,SCENTRY,EXTRACT,GPTRLOC,MINUSTWO
           B    PUTSCENTRY
           DCW  '0000000001I'
ONE       EQU  *-1+X1
           DCW  '0000000001I'

ZERO      EQU  *-1+X1
           DCW  '00000000KI'
MINUSTWO  EQU  *-1+X1
ENTER     EQU  ONE
EXTRACT   EQU  ZERO
           DCW  '.........9'
EOSCROCK  EQU  *-1+X1
           DCW  ' B2'
ENDVAR    EQU  *-1+X1
           FIN
           END
```

SECTION VI

SYSTEM MESSAGES

INTRODUCTION

The C-10 system consists of many separate routines. For purposes of classification, these routines are grouped under ten headings - Arithmetic, Control, Disk Allocation, Input-Output, PROFILE, P-stack, STEP, STEP Compiler, Stream Control and TAP. Some of these headings may also contain subheadings. For instance, the Control routines consist of routines which comprise the C-10 Editor, Executive and Loader - plus many other routines.

Following each message below is an indication of what routine it came from, and the heading under which the routine may be found. In some instances, the subheadings are included in parentheses. Many messages come from more than one routine. For some, a reference is given to the page(s) in Volume I or II* where the situation which may have given rise to the particular message is explained.

Example:

FORMAT ERROR.CHECK PARENTHESIS.
*
FDESC   PROFILE(TRANSLATION)
*   Page 38

STRUCTURE   PROFILE(TRANSLATION)
                                        Page 38

---

*Except for entries marked Volume II, all page numbers refer to Volume I.

The message FORMAT ERROR.CHECK PARENTHESIS occurs in the
routines: FDESC and STRUCTURE. Both of these routines are part
of the PROFILE translator. Reference is given to The CREATE
DICTIONARY Statement (ESD-TR-66-653, Volume I, Section III).

A dotted line in parenthesis represents variable infor-
mation in the message filled in at the time of the error by the
program detecting the error. When such an insertion begins the
message, the message is listed alphabetically by the word follow-
ing the insertion. An 'I' preceding the heading indicates a
strictly internal error which should be referred to a system
programmer.

MESSAGES

A-ARITHMETIC

A GROUP IS NOT IN THE DICTIONARY.   Page 38
*
SUBLOOKUP   PROFILE(TRANSLATION)

APOSTROPHE IS MISSING.   Pages 186, 104
*
TCHAT   PROFILE(TRANSLATION)

APOSTROPHE IS MISSING. PROCESSING WILL CONTINUE.   Pages 186, 119
*
HEADING   PROFILE(TRANSLATION)

ARITHMETIC ERROR-ZERO WITH NEGATIVE OR ZERO EXPONENT
*
NPOWER   ARITHMETIC

ARITHMETIC ERROR-ARGUMENT CONVERTS OUT OF INTEGRAL RANGE.
*   Pages 183, 184
FIX   ARITHMETIC

ARITHMETIC ERROR-DIVIDE BY ZERO.
*
FDIV   ARITHMETIC

ARITHMETIC ERROR- IMPROPER ARGUMENT
*
ADD   ARITHMETIC
*
ADD1   ARITHMETIC
*
ARCSIN   ARITHMETIC
*
ARITHEQ   ARITHMETIC
*
DIV   ARITHMETIC
*
EXP   ARITHMETIC
*
FLOAT   ARITHMETIC
*
MAX   ARITHMETIC
*
MIN   ARITHMETIC
*
MOD   ARITHMETIC
*
MUL   ARITHMETIC
*
PLUS   ARITHMETIC
*
SUB   ARITHMETIC
*
SUB1   ARITHMETIC
*
TIMES   ARITHMETIC

ARITHMETIC ERROR- INTEGER RESULT OUT OF RANGE.
*
ADD   ARITHMETIC   Pages 183, 184

*
MUL   ARITHMETIC   Pages 183, 184

*
SUB   ARITHMETIC
                                             Pages 183, 184

ARITHMETIC ERROR- NEGATIVE ARGUMENT.
*
SQRT   ARITHMETIC

ARITHMETIC ERROR- RESULT OUT OF RANGE.
*
FADD   ARITHMETIC   Pages 183, 184

*
FDIV   ARITHMETIC
                                                        Pages 183, 184
*
FMUL   ARITHMETIC
                                                        Pages 183, 184
*
FSUB   ARITHMETIC
                                                        Pages 183, 184
*
NPOWER   ARITHMETIC
                                                        Pages 183, 184

ATOM INPUT IS ILLEGAL TYPE- NOT I, H, P, OR L. ATOM IS IGNORED.
*
QLOTSI   I-PROFILE(EXECUTION)   **
                                                        Page 183

BACKSPACE OPERATION EXCEEDS AVAILABLE LOOKBACK.
*
INFIELD   PROFILE(TRANSLATION)
*
OUTFIELD   PROFILE(TRANSLATION)

BAD ARGUMENT (---), CHANGES STOPPED HERE.
*
CHANGPROG   CONTROL

BAD TAPE ON UNIT (---). NO RECOVERY.
*
SORT   PROFILE(EXECUTION)

BUFFER IS NOT AVAILABLE AT THIS TIME.
*
SPRINT1   PROFILE(EXECUTION)

CALLING ROUTINE (---) WHICH HAS NOT BEEN LOADED.
*
START   I-CONTROL

CARD-DICTIONARY

CARD READER NOT READY OR EMPTY. PLEASE SERVICE IT.
●
EDITOR   CONTROL(EDITOR)
●
PRCARD   CONTROL

DATA CHECK ON CARD READER. PRESS START WHEN READY.
●
EDITOR   CONTROL(EDITOR)

●
PRCARD   CONTROL

DATA CHECK ON CHANNEL (---).
●
CHANGPRCG   CONTROL

DATA CHECK OR TIMING CHECK ON PRINTER.
●
SPRINT   PROFILE(EXECUTION)

DATA CHECK. RE-ENTER UNIT RECORD.
●
CHANGPRCG   CONTROL

●
EDITOR   CONTROL

●
PRCARD   CONTROL

DATA DOES NOT AGREE WITH DICTIONARY DESCRIPTION. GIVEN (---), EXPECTS (---).
●
PACK   PROFILE(EXECUTION)

DELETE PROCEDURE DOES NOT EXIST.
●
DELST   PROFILE(TRANSLATION)

(---) DICTIONARY DOES NOT EXIST.
●
PDICT   PROFILE(TRANSLATION)
●

126

(---) DICTICNARY IS EMPTY.
*
CREATE   PROFILE(TRANSLATION)
*
DELST   PROFILE(TRANSLATION)

DICTICNARY IS MISSING.
*
RPLST   PROFILE(TRANSLATION)

DICTIONARY NAME IS MISSING.
*
CREATE   PROFILE(TRANSLATION)   Page 38

DICTIONARY POINTER DOES NOT EXIST
*
DELST   PROFILE(TRANSLATION)
*
DELSUB   PROFILE(TRANSLATION)

DID NOT FIND PARENTHESIS.
*
FUAS   TAP

(---) (---) DOES NOT EXIST.
*
SUPERCLA   I-PROFILE(EXECUTION)

END STATEMENT IS MISSING.
*
CONTROL   PROFILE(TRANSLATION)   Page 103

ENTER CHANNEL, UNIT OF OUTPUT TAPE.
*
SYSTAPE   CCNTROL

ENTER NUM, KEY, AND MESSAGE.
*
CHANGTEAR   CCNTROL

ENTER PROG TO REPLACE AND NEW NAME.
*
CHANGPROG   CONTROL

ENTER STARTING SUBROUTINE.
*
START   CONTROL   Vol. II, Page 36

ERROR IN DICTIONARY DESCRIPTION. NEITHER FIXED NOR VARIABLE TYPE
*
PACK   PROFILE (EXECUTION)

FILE NAME IS MISSING.
*

COMPLETE   PROFILE (TRANSLATION)   Page 61
*
DELST   PROFILE (TRANSLATION)   Page 50
*
INSRT   PROFILE (TRANSLATION)   Page 46
*
PDICT   PROFILE (TRANSLATION)   Page 52
*
RPLST   PROFILE (TRANSLATION)   Page 48
*
TRENAME   PROFILE (TRANSLATION)   Page 44

FILE NAME MUST BE SPECIFIED IN A WRITE STATEMENT.
*
PWSUBSET   PROFILE (TRANSLATION)   Page 113

FORMAT ERROR
*
RF'LST   PROFILE (TRANSLATION)   Page 48
*
TRENAME   PROFILE (TRANSLATION)   Page 44
*
WHOME   CONTROL (LOADER)

FORMAT-HANDS

FORMAT ERROR (---). CHANGES STOPPED HERE.
•
CHANGPRCG   CONTROL

FORMAT ERROR. CHECK PARENTHESIS.
•
FDESC   PROFILE(TRANSLATION)

•
STRUCTURE   PROFILE(TRANSLATION)   Page 38

                                                     Page 38

FORMAT ERROR. PART OF STATEMENT MISSING.
•
BOOLEAN   PROFILE(TRANSLATION)

                                                     Page 90

GIVEN IMPROPER ARG
•
EXTRACT   PROFILE(EXECUTICN)
•
PPGLOB   STEP COMPILER
•
TTSUBLCCKUP   PROFILE(TRANSLATION)

GIVEN IMPROPER ARG. ELEMENT (---) DOES NOT EXIST.
•
TT   PROFILE(TRANSLATION)

GROUP NAME CN LEFT HAND SIDE OF CHANGE STATEMENT IS NCT ALLOWED.
•
TCHANGE   PROFILE(TRANSLATION)
                                                     Page 110

GROUP NAMES CN LEFT HAND SIDE OF ASSIGNMENT STATEMENT ARE NOT ALLOWED.
•
ASSIGNMENT   PROFILE(TRANSLATION)
                                                     Page 62

 HANDS OFF THE INCUIRY REQUEST BUTTON
•
READ   INPUT/OUTPUT

129

HE-F-L-LP
•
SUPERTRACE   I-CONTROL

ILLEGAL A-STACK REQUEST INVOLVING ORDERING OF FILE DATA
•
CLEANSORT   I-PROFILE(EXECUTION)

ILLEGAL DEVICE NAME.
•
TINPUT   PROFILE(TRANSLATION)

•
TOUTPUT   PROFILE(TRANSLATION)

ILLEGAL EDIT COMMAND. EDITING TERMINATED.
•
EDITCOMC   CONTROL

ILLEGAL FIELD OPERATION.
•
TFIELD   PROFILE(TRANSLATION)

ILLEGAL FIELD OPERATION AFTER N OR NOT.
•
TFIELD   PROFILE(TRANSLATION)

ILLEGAL IOCS PARAMETERS. ARG1 = (---). ARG2 = (---). ARG3 = (---). I/O NOT
EXECUTED.
•
IOMES   INPUT/OUTPUT

ILLEGAL NAME IN INPUT STATEMENT.
•
TINPUT   PROFILE(TRANSLATION)

ILLEGAL-INCORRECT

ILLEGAL SORT SPECIFICATIONS.
*
SORT   PROFILE(TRANSLATION)
                                        Page 111

ILLEGAL UNIT NUMB FOR QIN OR QOUT. N=(---).
*
IOMES   INPUT/OUTPUT

IMPROPER BOOLEAN FOLLOWING-IF- IN SUBSET STATEMENT
*
GSDLIST   PROFILE(TRANSLATION)
                                        Page 90
*
PSDLIST   PROFILE(TRANSLATION)
                                        Page 90

IMPROPER EXPRESSION FOLLOWING = IN SUBSET STATEMENT.
*
PSDLIST   PROFILE(TRANSLATION)
                                        Page 88

IMPROPER GLOBAL CONTENTS FOR SCLAMG. CROCK CONTENTS ARE (---). TYPE CODE
IS (---).
*
UNSUPERC   I-PROFILE(EXECUTION)

IMPROPER GO FORM.
*
EVAL   STEP
                                        Vol. II, p. 97

INCORRECT DATA - ALPHANUMERIC TYPE WITH FIXED LENGTH ZERO.
*
PACK   PROFILE(EXECUTION)

INCORRECT DATA - BLANK LITERAL.
*
PACK   PROFILE(EXECUTION)

INCORRECT FORMAT
*
CSACTOR   TAP
*
CSFIN   TAP

INCORRECT FORMAT. CANNOT MAKE DEFINITION
*
DACTOR   TAP

INCORRECT FORMAT. UNEXPECTED RIGHT PARENTHESIS FOUND.
*
SETUPT   TAP

INCORRECT PUNCTUATION.
*
RPLST   PROFILE (TRANSLATION)   Page 48
*
TRETURN   PROFILE (TRANSLATION)   Page 107

INPUT DATA EXHAUSTED. LOOKING FOR MORE.
*
INFIELD   PROFILE (TRANSLATION)

INQUIRY CANCEL.
*
EDITOR   CONTROL (EDITOR)

INQUIRY CANCEL. REPEAT INPUT.
*
IOMES   INPUT/OUTPUT

INPUT OR OUTPUT IS MISSING.
*
SPACING   PROFILE (TRANSLATION)

                                         Pages 79, 95

INVALID ARGUMENT.
*
INFIELD   PROFILE (TRANSLATION)
*
OUTFIELD   PROFILE (TRANSLATION)

INVALID BOOLEAN WITHIN CONDITIONAL.
*
CONDITION   PROFILE(TRANSLATION)

Page 90

INVALID CALL TO PROCEDURE OR UNDECIPHERABLE EXPRESSION.
*
DO   PROFILE(TRANSLATION)
Pages 88, 109

INVALID CROCK. EXPECTS A STREAM POINTER.
*
INFIELD   PROFILE(TRANSLATION)
*
OUTFIELD   PROFILE(TRANSLATION)

INVALID EXPRESSION
*
DEFLOG1   PROFILE(TRANSLATION)
Page 88
*
TCHANGE   PROFILE(TRANSLATION)
Page 88
*
EXPRESSIO   PROFILE(TRANSLATION)
Page 88
*
SUBBOOL   PROFILE(TRANSLATION)
Page 88

INVALID EXPRESSION. FOR PAGE. PROCESSING WILL CONTINUE.
*
HEADING   PROFILE(TRANSLATION)
Page 88

INVALID EXPRESSION USED AS INPUT ARGUMENT TO PROCEDURE OR FUNCTION.
*
DO   PROFILE(TRANSLATION)
Page 88

INVALID KEY WORD FOR TERSE DEFINITION
*
SETUPT   TAP

133

INVALID KEYWORD. WILL CONTINUE.
*
HEADING   PROFILE (TRANSLATION)

INVALID NAME IN AN OUTPUT STATEMENT
*
TOUTPUT   PROFILE (TRANSLATION)   Page 93

INVALID NAME IN CHANGE STATEMENT.
*
TCHANGE   PROFILE (TRANSLATION)
                                                          Pages 35-37, 110

INVALID NAME IN VARIABLE DECLARATION
*
VDECLARAT   PROFILE (TRANSLATION)
                                                          Pages 35-37, 72

INVALID PROPERTY-NAME IN SORT STATEMENT.
*
TSORT   PROFILE (TRANSLATION)
                                                          Pages 35-37, 111

INVALID PROPERTY-NAME IN SUBSET STATEMENT.
*
PSDLIST   PROFILE (TRANSLATION)
                                                          Pages 35-37, 113-119

INVALID PSTACK VALUE
*
GETATOM   TAP

INVALID RELATION.
*
SUBBOOL   PROFILE (TRANSLATION)   Page 90

INVALID SIMPLE GROUP TYPE
*
GETATOM   TAP

INVALID STATEMENT FOLLOWING ELSE WITHIN CONDITIONAL.
*
CONDITICN   PROFILE(TRANSLATION)

INVALID STATEMENT WITHIN CONDITIONAL.
*
CONDITICN   PROFILE(TRANSLATION)

INVALID Z-POINTER
*
GETAPATCM
*
SETBIT   TAP

(---) IS NOT DEFINED IN THE CURRENT P-STACK.
*
NILLIST   I-PROFILE(EXECUTION)
*
CLEANUP   I-PROFILE(EXECUTION)

(---) IS NOT IN (---) DICTIONARY.
*
CREATE   PROFILE(TRANSLATION)

*
DELST   PROFILE(TRANSLATION)

*
INSRT   PROFILE(TRANSLATION)

KEYWORD -BUT- IS MISSING. PROCESSING WILL CONTINUE.
*
RPLST   PROFILE(TRANSLATION)

KEYWORD -FOR- IS MISSING.
*
CREATE   PROFILE(TRANSLATION)
*
DEFLOG1   PROFILE(TRANSLATION)

KEYWORD -LEOB, LEOP, OR LEOI- IS MISSING.
*
DEFLOG1   PROFILE(TRANSLATION)

Pages 77, 95

KEYWORD -INPUT OR OUTPUT- IS MISSING.
*
DEFLOG1   PROFILE(TRANSLATION)

Pages 77, 95

KEYWORD -OR- IS MISSING. PROCESSING WILL CONTINUE.
*
INSRT   PROFILE(TRANSLATION)

KEYWORD -TO- MISSING IN CHANGE STATEMENT.   Page 110
*
TCHANGE   PROFILE(TRANSLATION)

LEGAL FIELD OPERATION ON LEFT-HAND SIDE OF ASSIGNMENT STATEMENT NOT FOLLOWED
IMMEDIATELY BY AN EQUAL SIGN (=).
*
ASSIGNMENT   PROFILE(TRANSLATION)

Page 98

LINE (---)-ILLEGAL CHARACTER USED IN INPUT - GROUP MARK, RECORD MARK OR TAPE
SEGMENT CHARACTER.
*
EDITSEG   CONTROL(EDITOR)

Page 181

LINE (---)- ILLEGAL NUMERIC FIELD. (---) WAS INTERPRETED AS (---).
*
EDITSEG   CONTROL(EDITOR)

Page 183, 184

LINE (---) TOO LONG. IGNORED.
*
EDITOR   CONTROL(EDITOR)

Page 172

LITERAL IS TOO LONG
*
GETATOM   TAP

NO GROUP NAME SPECIFIED.
*
TCLOSE   PROFILE(TRANSLATION)

NO GROUP NAME SPECIFIED. PROPERTY NAME SPECIFIED INSTEAD OF GROUP NAME.
*
TCLOSE   PROFILE(TRANSLATION)

NON-GROUP NAMES NOT ALLOWED IN WRITE STATEMENT.
*
WRITE   PROFILE(TRANSLATION)

NC STATEMENT FOLLOWING LABEL.
*
LABELED   PROFILE(TRANSLATION)

(---) NCT DEFINED.
*
XLSTEP   STEP COMPILER

NOTE - THE NAME (---) WAS ALREADY DEFINED.
*
CHANGPRCG   CCNTROL

(---) CN CHANNEL (---) NOT READY. READY IT.
*
IOMES   INPUT/OUTPUT

ONLY GROUP NAMES MAY BE USED IN REAC CR PROCESS STATEMENTS.
*
TREAD   PROFILE(TRANSLATION)

ONLY VARIABLES DECLARED IN A VARIABLE DECLARATION STATEMENT MAY BE USED AS
OUTPUT ARGUMENTS TO A PROCEDURE OR FUNCTION.
*
DO   PROFILE (TRANSLATION)

NUMBER OF ARGUMENTS EQUAL ZERO. ARGUMENTS ARE EXPECTED.
*
EVAL   STEP

NUMBER OF ARGUMENTS AND NUMBER OF INPUT VARIABLES IN A PROCEDURE DO NOT AGREE.
*
E VAL   STEP

OVER 10 ARGS.
*
EVAL   STEP

PARENT GROUP (---) FOR OWNID (---) NOT CURRENTLY OPEN.
*
POSITION   PROFILE (TRANSLATION)

PARENTHESIS PROBLEM IN SUBSET STATEMENT.
*
GSDLIST   PROFILE (TRANSLATION)
*
PWSUBSET   PROFILE (TRANSLATION)

PLEASE REPORT . . .
*
TERROR   I-PROFILE (SPECIAL TOOLS)

(---) POINTER DOES NOT EXIST.
*
INSRT   PROFILE (TRANSLATION)

PRINTER NOT READY. PLEASE SERVICE IT.
*
PRCARD   CONTROL
*
PRDIR   CONTROL

PROCEDURE (---) HAS BEEN DEFINED.
*
DEFINE   STEP
*
DEFINEFM   STEP

PROCEDURE (---) WAS FORMERLY A RELOCATABLE PROGRAM. THE PROGRAM WILL BE THE
PERMANENT ENTRY.
*
DEFINE   STEP
*
DEFINEFM   STEP

PROPERTY NAME IS MISSING.
*
DELST   PROFILE(TRANSLATION)

*
RPLST   PROFILE(TRANSLATION)

PROPERTY NAME MENTIONED BEFORE OPENING FILE.
*
TSORT   PROFILE(TRANSLATION)

PROPERTY NAME MENTICNED WITHOUT SUFFICIENT PRECEDING READ STATEMENTS.
*
TSORT   PROFILE(TRANSLATION)

PROPERTY NAME CN LEFT-HAND SIDE OF ASSIGNMENT STATEMENT WHOSE PARENT GROUP
HAS NOT BEEN MENTIONED IN A WRITE STATEMENT.
*
ASSIGNMNT   PROFILE(TRANSLATION)

PROPERTY NAME SPECIFIED INSTEAD OF GROUP NAME.
*
TCLOSE   PROFILE (TRANSLATION)   Pages 35, 58

(---) READY.
*
EDITOR   CONTROL (EDITOR)

READY BUFFER UNITS 1, 2, 3, 4. OPEN R/W FILE LIST.
*
SUPERLOC   PROFILE (TRANSLATION)

READY PUNCH
*
WHOME   CONTROL (LOADER)

RELOAD FROMTAPE .
*
C-10 LOADER   CONTROL (LOADER)

RETURN TO COLINGO EXEC.
*
START   CONTROL

REWIND IS NECESSARY.
*
TINPUT   PROFILE (TRANSLATION)
                                            Page 75

RIGHT HAND SIDE OF ASSIGNMENT STATEMENT NOT A LEGAL EXPRESSION.
*
ASSIGNMENT   PROFILE (TRANSLATION)
                                            Page 88
*
TFIELD   PROFILE (TRANSLATION)
                                            Page 88

SEARCH FOR OWNID (---) IN SUPERCLA STACK YIELDS UNDEFINED.
*
POSITION   PROFILE (TRANSLATION)

SECOND DICTICNARY NAME IS MISSING.
*
CREATE   PROFILE(TRANSLATION)

SEMI-CCLON IS MISSING AT END OF BOOLEAN WITHIN CONDITICNAL.
*
CONDITICN   PROFILE(TRANSLATION)

SFMI-CCLON IS MISSING AT END OF STATEMENT. PRCCESSING WILL CONTINUE.
*
CCNTROL   PROFILE(TRANSLATION)

SEMT-CCLON IS MISSING AT END OF STATEMENT WITHIN CONDITIONAL.
*
CONDITICN   PROFILE(TRANSLATION)

SIGN - FOLLCWED BY INPUT INSTEAC OF NUMBER. SIGN IS IGNORED.
*
QLOTSI   PROFILE(TRANSLATION)

SORRY-THERE IS NO LARGE BUFFER AVAILABLE. AN ENDC10 MUST BE GIVEN AFTER THIS
RUN TO MAKE THE DEFINE PERMANENT.
*
DEFINE   STEP
*
DEFINEFN   STEP

STATEMENT FCRMAT ERROR. PROCESSING WILL CCNTINUE.
*
CONTRCL   PROFILE(TRANSLATION)

SYNTAX ERROR IN SORT STATEMENT.
*
TSCRT   PROFILE(TRANSLATION)

SYSTEM ERROR. PUNTING ADVISED.
*
EDITCOMP   I-CCNTROL(EDITOR)

THE ARGUMENT TO DEFINE IS NIL, NOTHING HAS BEEN DEFINED.
•
DEFINE   STEP
•
DEFINEFN   STEP

THE STRUCTURE OF THIS DEFINE OPERATION IS BAD. IT HAS NOT BEEN DEFINED. (---)
•
DEFINE   STEP
•
DEFINEFN   STEP

THERE IS NO ROOM IN THE SUBROUTINE DIRECTORY. PROCEDURE (---) HAS NOT BEEN
LOADED.
•
FINDPROC   STEP

TOO MANY ARGUMENTS FOR TERSE.
•
SETUPT   TAP

TOO MANY ALT. BLANKS. LINE (---) IGNORED.
•
EDITOR   CONTROL(EDITOR)

Page 174

UNABLE TO FIND STREAM ASSOCIATED WITH (---).
•
PACK   PROFILE(EXECUTION)

UNABLE TO LOCATE (---). CHANGES STOPPED HERE.
•
CHANGPROC   CONTROL

UNABLE TO PROCESS MORE THAN 1 PAGE AT PRESENT TIME. ANYMORE ENTRIES WILL
BE REJECTED.
•
SPRINT1   PROFILE(EXECUTION)

UNACCEPTABLE LABEL IN A GO-TO STATEMENT.
*
GOTO   PROFILE (TRANSLATION)   Page 101

UNDEFINED FORM-FUNCTION
*
EVAL   STEP

UNMATCHED LEFT PARENTHESIS IN ARGUMENT LIST OF PROCEDURE OR FUNCTION .
*
DO   PROFILE (TRANSLATION)
                                                   Page 109

UNMATCHED TERMINATE STATEMENT.
*
TERMINATE   PROFILE (TRANSLATION)   Page 123

UNNECESSARY FILE-NAME SPECIFIED.
*
PWSUBSET   PROFILE (TRANSLATION)
                                                   Pages 113-119

UNPAIRED BRACKETS.
*
SIMBOOL   PROFILE (TRANSLATION)

UNRECOGNIZABLE FILE-NAME IN SUBSET STATEMENT.
*
PWSUBSET   PROFILE (TRANSLATION)
                                                   Pages 113-119

UNRECOGNIZABLE GROUP-NAME IN SORT STATEMENT.
*
TSORT   PROFILE(TRANSLATION)

Page 111

UNRECOGNIZABLE NAME IN READ OR PROCESS STATEMENT.
*
TPROCESS   PROFILE(TRANSLATION)

Page 120
*
TREAD   PROFILE(TRANSLATION)

Page 55

WRONG LENGTH RECORD TO PRINTER.
*
SPRINT   PROFILE(EXECUTION)

Z-STACK EMPTY OR NON-EXISTENT.
*
PPGLOB   STEP COMPILER

APPENDIX I

SUMMARY OF LOADING INSTRUCTIONS

Summary of Loading Instructions	References	
	From Card	From Tape
1. Clear core, rewind tape (if tape is being used)		
2. Ready proper deck setup in card reader.	pp 11, 12, 13	p 15
3. Enter and execute bootstrap instruction.	p 14	p 16
4. Make necessary changes in configuration.	pp 17-28	pp 17-28
5. Initialize.	pp 32-36	p 36

Summary of Operating Instructions

1. If loading has just been completed, skip to step 4.		
2. Ready bootstrap card in card reader.	p 38	
3. Enter and execute bootstrap instructions.	p 38	
4. If STEP - PROFILE are to be used, type 237.	pp 38, 39	

145

APPENDIX II

C-10   1301 - 1311 DISK ADDRESSES

The process for converting 1301 disk addresses to 1311 disk addresses is described as follows.

First a relative 1301 disk address is computed with the formula:

$$A_2M_2T_2H_2 = A_1M_1T_1H_1 - BASE$$

where

A_1 = original access number

M_1 = original module number

T_1 = original track number

H_1 = original HA2 number

A_2 = relative access number

M_2 = relative module number

T_2 = relative track number

H_2 = relative HA2 number

BASE = lowest 1301 disk control field to be converted to 1311 control field.

Then the 1311 section address ,S, is computed with the following formula (using integer division):

$$S = X + 2(X/198)$$

where

$$X = 6(4(T_2)+H_2)$$

APPENDIX III

STEP IN BACKUS NORMAL FORM

< name >   :: = defined in ESD-TR-66-653, Volume I, Section V
                    (The Editor)

< literal > :: = defined in ESD-TR-66-653, Volume I, Section V
                    (The Editor)

< integer > :: = defined in ESD-TR-66-653, Volume I, Section V
                    (The Editor)

< floating point number > :: = defined in ESD-TR-66-653, Volume I,
                                Section V (The Editor)

< number > :: = < floating point number > | < integer >

< expression > :: = < name > | < number > | < literal > |
                      < procedure form > |
                      < prog form > |
                      < if form > |
                      < cond form > |
                      < go form > |
                      < return form > |
                      < assign form > |
                      < quote form > |
                      < define form > |
                      < atom form >

< procedure form > :: = (< procedure name > < expression list >) |
                          (< autonomous procedure name >
                            < expression list >)

< expression list > :: = ( ) | (< stripped expression list >)

< stripped expression list > :: = < expression > | < expression >
                                                  < stripped expression list >

< autonomous procedure name > :: = (FUNCTION < name list > < expression >) |
                                   (FORM < name list > < expression >)

< prog form > :: = (PROG < name list > < expression list >)

< cond form > :: = (COND < sequent list >)

< sequent list > :: = (< sequent >) | (< sequent >) < sequent list >

< sequent > :: = < expression > < expression >

< if form > :: = (IF < sequent >)

< go form > :: = (GO < name >)

< return form > :: = (RETURN < expression >)

< assign form > :: = (ASSIGN < name > < expression >)

< quote form > :: = (QUOTE < number >) | (QUOTE < list >)| QUOTE < number > |
                                           QUOTE < list >

< atom form > :: = (ATOM < name >) | (ATOM < number >) | ATOM < name > |
                                           ATOM < number >

< list > :: = ( ) | (< sequence >)

< sequence > :: = < sequence element > | < sequence element > < sequence >

< sequence element > :: = < name > | < number > | < list >

< name list > :: = ( ) | (< name sequence >)

< name sequence > :: = < name > | < name > < name sequence >

< define form > :: = (DEFINE < definition sequence >)

< definition sequence > :: = < definition > | < definition >
                                         < definition sequence >

< definition > :: = (< procedure name > (FUNCTION < name list >
                < expression >)) | (< procedure name >
                            (FORM < name list > < expression >))

148

INDEX

This index to the COLINGO C-10 USERS' MANUAL contains references to both volumes.

A reference, then, consists of the volume number followed by the page numbers.

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1 ORIGINATING ACTIVITY (Corporate author)	2a. REPORT SECURITY CLASSIFICATION
The MITRE Corporation Bedford, Massachusetts	UNCLASSIFIED
	2b. GROUP N/A

3. REPORT TITLE

COLINGO C-10 USERS' MANUAL -- VOLUME II

4 DESCRIPTIVE NOTES (Type of report and inclusive dates)

N/A

5. AUTHOR(S) (First name, middle initial, last name)

COLINGO Project

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
May 1968	162	0

8a. CONTRACT OR GRANT NO. AF 19(628)-9165	9a. ORIGINATOR'S REPORT NUMBER(S) ESD-TR-66-653, Vol. II
b. PROJECT NO. 512V	
c.	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report) MTR-35
d.	

10. DISTRIBUTION STATEMENT

This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES N/A	12. SPONSORING MILITARY ACTIVITY Air Force Command and Management Systems Division, Deputy for Command Systems, Electronic Systems Division, L. G. Hanscom Field, Bedford, Mass.

13. ABSTRACT

The COLINGO C-10 Users' Manual, a combination of tutorial and reference material, is presented in two volumes. This volume contains information on machine configurations, procedures for operating and loading the system, a description of the C-10 general purpose macro facility (terses and actors), a guide to the STEP language, a set of instructions for preparing machine procedures, and a list of system error messages.

DD FORM 1473
1 NOV 65

14	KEY WORDS	LINK A		LINK B		LINK C	
		ROLE	WT	ROLE	WT	ROLE	WT
	PROFILE						
	PROGRAMS						
	STATEMENTS						